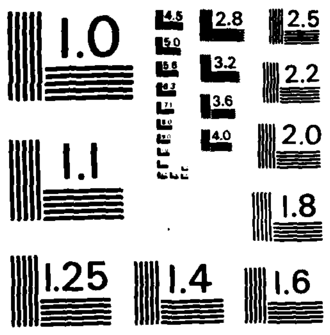
AD-A139 424   FORJR: AN IMPLEMENTATION OF BADJR USING FORTH AND Z80   1/2
ASSEMBLY LANGUAGE(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH W M EDMONSON 1983
UNCLASSIFIED AFIT/CI/NR-83-87T   F/G 9/2   NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

ADA139424

1

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR-83-87T	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FORJR: An Implementation of BADJR Using FORTH and Z80 Assembly Language		5. TYPE OF REPORT & PERIOD COVERED THESIS/~~DISSERTATION~~ 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William M. Edmonson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: Wright State University		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433		12. REPORT DATE 1983 13. NUMBER OF PAGES 178
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASS 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

[illegible]

[illegible]

LYNN E. WOLAVER
Dean for Research and
Professional Development
AFIT, Wright-Patterson AFB OH

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

S [illegible] D
A

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASS

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

[illegible]

[illegible] BASIC [illegible] FORTH [illegible]

[illegible]

[illegible] BASIC [illegible] The [illegible] programming language, FORTH, [illegible] assembly language modules and can be run on [illegible] under the UNIX Operating System. [illegible] implements the attribute grammar of BASIC. The assembly language portion of FORTH [illegible] The FORTH environment [illegible] the [illegible] assembly language module [illegible] FORTH. [illegible] attribute grammar [illegible]

[illegible]

[illegible]

AFIT/CI/NR 83-87T

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to ascertain the value and/or contribution of research accomplished by students or faculty of the Air Force Institute of Technology (ATC). It would be greatly appreciated if you would complete the following questionnaire and return it to:

AFIT/NR
Wright-Patterson AFB OH 45433

RESEARCH TITLE: FORJR: An Implementation of BADJR Using FORTH and Z80 Assembly Language

AUTHOR: William M. Edmonson

RESEARCH ASSESSMENT QUESTIONS:

1. Did this research contribute to a current Air Force project?

( ) a. YES ( ) b. NO

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not?

( ) a. YES ( ) b. NO

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency achieved/received by virtue of AFIT performing the research. Can you estimate what this research would have cost if it had been accomplished under contract or if it had been done in-house in terms of manpower and/or dollars?

( ) a. MAN-YEARS ________ ( ) b. $________

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3. above), what is your estimate of its significance?

( ) a. HIGHLY SIGNIFICANT ( ) b. SIGNIFICANT ( ) c. SLIGHTLY SIGNIFICANT ( ) d. OF NO SIGNIFICANCE

5. AFIT welcomes any further comments you may have on the above questions, or any additional details concerning the current application, future potential, or other value of this research. Please use the bottom part of this questionnaire for your statement(s).

NAME	GRADE	POSITION
ORGANIZATION	LOCATION	

STATEMENT(s):

ABSTRACT

Edmonson, William M., M.S., Department of Computer Science, Wright State University, 1983. FORJR: An Implementation of BADJR Using FORTH and Z80 Assembly Language.

The FORJR project implements a system to provide an interactive BADJR functional programming machine. The interactive programming language, FORTH, is combined with Z80 assembly language modules and can be run on Z80-based systems under the CP/M Operating System. A frame-stack mechanism implements the attribute grammer of BADJR. The assembly language portion of FORJR was developed independently of this project, but is modified to provide an interface with FORTH. The FORTH environment set up calls to the specific assembly language modules which manipulate attribute storage areas. Upon completion of specified tasks, execution control is returned to FORTH. Special attention is directed at storage management of FORJR, including details of attribute passing, garbage collection and compaction.

Examples of FORJR programs are provided including explanations and illustrations of simple and recursive FORJR calls.

[illegible]

[illegible] Robert [illegible]

[illegible]

[illegible]

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

1.0 PRIMARY OBJECTIVES

The primary objective of the FORJR project was to implement an interactive BADJR functional programming machine using FORTH and Z80 assembly language modules. BADJR is a functional language currently under research and development by Computer Science Department faculty and students of the FLITE Project at Wright State University, Dayton, Ohio. The functional specifications for the FORJR machine are based upon the BADJR Report [DIXO83].

FORJR, as the name implies, combined the interactive facilities of FORTH with a BADJR functional language machine. The BADJR machine used in this project was developed independently in Richard Franklin's "ZBADJR: An Implementation of the BADJR Machine in Z80 Assembly Language" [FRAN83]. Certain modules of the ZBADJR code were modified to permit smooth transitions to and from the FORTH environment. An assembly language interface was developed to protect the FORTH environment and to set up the appropriate calls to ZBADJR routines.

FORJR is designed to run on any Z80-based system using the CP/M Operating System. The emphasis the project places on the interactive facilities of FORJR coincides with the increasing interest in using FORTH as a teaching tool at Wright State. Students already knowledgeable in FORTH should adapt readily to experimenting with functional programming in FORJR.

Examples of FORJR programming have beer provided, ranging from simple, single-line entries t(complex, recursive routines. However, as with other ‑ ɜractive systems, hands-on experimentation with FORJR proved to be the best research method.

Data object representation in FORJR closely resembles the structure used by Sloan [SLOA83]. The advantage FORJR has over other implementations is that the storage areas used to hold data objects can be examined periodically between FORJR function calls. This feature permits the user to see direct results on the data objects and storage areas between FORJR function calls.

Section II describes the FORJR machine environment. and the linking convention of the FORTH and ZBADJR files.

Section III describes the FORJR attribute and data object representation. In addition, storage management procedures are discussed including garbage collection and storage compaction.

Section IV details the syntax of FORJR instructions. Examples of simple FORJR functions calls are included. Section V concludes the FORJR Project discussion and includes recommendations for future research.

II. FORJR MACHINE ENVIRONMENT

1.0 INTRODUCTION

The ZBADJR system designed by Franklin provides a good system for studying functional programming. For the most part, ZBADJR models the BADJR machine as discussed in the original BADJR report. However, ZBADJR has a major limitation in that all ZBADJR user programs must be written, compiled, and linked in Z80 assembly language. This task does not lend itself to experimentation because of the time consumming task of writing test programs even for simple tests. FORJR circumvents this problem by combining the power of the ZBADJR assembly language modules with the ease of use of FORTH interactive programming.

2. FORTH AND ZBADJR INTERFACE

The ZBADJR source programs made extensive use of macro calls. The original system consisted of over 80 separate macros that resembled BADJR functions. The majority of these macros contained multiple instructions including additional macro calls. These macros manipulated the ZBADJR data storage areas by calls to specific Z80 assembly language routines. The basic design of FORJR was to establish an interface between FORTH and ZBADJR and devise methods to emulate the macro calls.

2.1 INTERFACING FORTH WITH ZBADJR

FORTH, through the use of assembly language instructions, has mechanisms by which other programs can be called, but the called programs must be in memory along with the FORTH system. A Z80 assembly language program was devised to act as an interface between FORTH and ZBADJR. This program has two functions. The first is to preserve the FORTH registers and return address to ensure a smooth transition from FORTH to ZBADJR and back to FORTH. The second function of the interface program involves using a jump table to invoke specific ZBADJR modules. The jump table will be discussed in the next section.

Because the ZBADJR programs and FORTH system must reside in memory together, special linking and loading conventions were needed to create a single executeable module. The Z80 interface program and ZBADJR modules are linked and loaded at location 9100H. Figure 1 shows the memory configuration of the FORJR system.

```
 100H +-----------------------+ <-----+
      |                       |       |
      |                       |       |
      |     FORTH System      |       |
      |                       |       |
      |                       |       |
9100H +-----------------------+       |
      |   Interface Program   |       + <---- FORJR
921FH +-----------------------+       |    (53k bytes)
      |                       |       |
      |                       |       |
      |    ZBADJR Modules     |       |
      |                       |       |
      |                       |       |
CF00H +-----------------------+ <-----+
```

FIGURE 1. FORJR Memory Configuration

The interface program provides the single entry point to the ZBADJR routines. When FORTH calls the interface program, location 9100H, the return address to FORTH is pushed onto the system stack. The interface program preserves the FORTH interpreter pointer (BC register) and the return pointer (IY register) in separate memory locations. The appropriate ZBADJR routine is then called via the jump table. After the ZBADJR routine executes, control returns to the interface program which restores the FORTH registers, pushes the FORTH return address onto the system stack and executes a return to FORTH.

2.2 ZBADJR JUMP TABLE

A jump table was created containing entries for each ZBADJR function. The jump table can be found in the first program of the Z-80 Source listings, Appendix C. All entries are 3-byte Z80 JUMP commands. Not all ZBADJR functions are currently installed in FORJR, so 3-byte entries were provided as place holders to permit future implementation. All valid entries in the jump table have a corresponding FORJR command. When a FORJR command is invoked, a value is placed onto the FORTH parameter stack. This value is then multiplied by three to provide a 3-byte offset into the jump table. FORTH then calls the interface routine. Since all ZBADJR routines execute a RETURN when complete, the interface routine pushes a return address onto the system stack prior to jumping to any ZBADJR routine. The interface routine then calculates the offset into the jump table where the appropriate ZBADJR routine is invoked.

Some FORJR routines need to pass parameters to the ZBADJR routines. The FORTH parameter stack, which is the same stack as the Z-80 system stack, is used for this purpose. The necessary parameters are pushed onto the FORTH stack prior to pushing the jump table index and calling the interface program. Any ZBADJR routine that returns parameters to FORTH reverses this process by pushing appropriate values onto the system stack prior to returning to the interface program.

III. FORJR DATA REPRESENTATION

1.0 ATTRIBUTES

The BADJR report defines data objects as attributes and describes three types: INHERITED, SYNTHESIZED, and LOCAL. BADJR uses these attributes to pass values between BADJR routines. All inherited attributes are defined, i.e. assigned a type and value prior to entry into a BADJR routine. Synthesized attributes are defined by BADJR routines and once defined may not be modified again. BADJR routines may also use local attributes that are defined and used only during that routine's execution.

FORJR attributes are represented by 2-byte hex numbers that are indices into a list of node descriptor blocks. As attributes are created, they are given a unique index value which is assigned in increasing order from 1 to N, where N is the maximum number of nodes permitted. FORJR currently has provisions for 256 nodes. Synthesized attributes are given an index without further defining the attribute type or value. When a FORJR routine is to define the synthesized attribute, the index of that attribute is passed to that FORJR routine along with relevant inherited attributes. The FORJR routine then assigns a type and value to the synthesized attribute.

When a synthesized attribute is defined, an address in the attribute's node description block is set to point to the location of an associated stringspace which contains the type and value of the attribute. More detailed explanations of the nodelist and stringspace areas can be found in paragraph 2, STORAGE MANAGEMENT.

1.1 ATTRIBUTE PASSING MECHANISM

FORJR uses a stack-oriented mechanism to pass attributes to other FORJR routines. Each routine operates on a 'frame' that contains attribute indices that the routine will use or define. All inherited attributes come from the pvevious frame. To rerieve atributes from the previous frame, the user must 'stack' the desired attributes onto the current frame. This is accomplished via the STKINH command. E.G. if you want the third attribute from the previous frame, enter:

```
3 STKINH
```

Frames are stacked in a data structure called the INHERITANCE STACK. FORJR uses attributes from the top most frame for all data manipulation. Therefore, before calling the FORJR routine, the current frame must contain all relavent attributes and in the order expected by the particular FORJR routine.

2.0 STORAGE MANAGEMENT

The BADJR Report described the properties of BADJR objects. FORJR follows the BADJR conventions except for one significant difference: numbers may be represented as fixed-point decimals as well as integers.

2.1 OBJECTS IN MEMORY

The Z-80 assembly language portion of FORJR contains the data storage areas used to hold frames and objects. The primary areas are the INHERITANCE STACK, NODELIST, and STRINGSPACE.

2.0.1 INHERITANCE STACK

Initially, the inheritance stack, sometimes called the frame stack, is set to zeros. As frames are created, a pointer to the floor of the old frame (OBAS or old-base-attribute-stack) is stored in the first word of the new frame which is the floor of the new frame (or BAS). The floor of the first frame contains the address of the inheritance stack ("ground") indicating it is the bottom frame on the stack. Figure 2 shows the inheritance stack with an initial frame containing the indicies of three attributes. (Note: The beginning address of the inheritance stack in Figure 2 is 98D0H.)

```
|------|------|------|------|------|--------
|  00  | 98D0 |  01  |  02  |  03  | 00 00 00 . . .
|      |      |      |      |      |
|      |ptr to| 1st  | 2nd  | 3rd  |
|      |"GND" | attr | attr | attr |
|      |      |index |index |index |
|  2b  |  2b  |  2b  |  2b  |  2b  |
|------|------|------|------|------|-------
```

FIGURE 2. Inheritance Stack With One Frame
(2b -> 2 bytes)

Figure 3 shows the inheritance stack with an additional frame stacked using two attributes from the first frame and two new attributes.

```
|----|----|----|----|----|----|----|----|----|----|
| 00 |98D0| 01 | 02 | 03 |98D2| 01 | 02 | 04 | 05 |
|    |    |    |    |    |    |    |    |    |    |
|    |GND |1st |2nd |3rd |1st |1st |2nd |4th |5th |
|    |ptr |attr|attr|attr|frm.|attr|attr|attr|attr|
|    |    |indx|indx|indx|ptr |indx|indx|indx|indx|
| 2b | 2b | 2b | 2b | 2b | 2b | 2b | 2b | 2b | 2b |
|----|----|----|----|----|----|----|----|----|----|
     |                   |                        |
     |    First Frame    |      Second Frame      |
```

FIGURE 3. Inheritance Stack With Two Frames

2.0.2 NODELIST

The attribute indices mentioned above are unique 2-byte indices into the nodelist. These attribute indices are allocated sequentially. The nodelist containing the attribute indices consists of 4-byte nodes. The first two bytes is an address field pointing to a stringspace representing a corresponding attribute. The third byte is a tag field and the forth byte is unused. The use of the address field is discussed below. An explanation of the tag field is in paragraph 2.3, GARBAGE COLLECTION AND STORAGE COMPACTION. Initially, all nodes are set to "avail", indicated by FFFFH. Figure 4. depicts the initial nodelist.

```
|----|---|---|----|---|---|----|---|---|
|FFFF| 0 | 0 |FFFF| 0 | 0 |FFFF| 0 | 0 |
|    |   |   |    |   |   |    |   |   |. . .
|ADDR|TAG|UNU|    |   |   |    |   |   |
| 2b |1b |1b |    |   |   |    |   |   |
|----|---|---|----|---|---|----|---|---|
|           |            |            |
|4-byte node|            |            |
```

FIGURE 4. Initial NodeList

In the event a synthesized attribute is allocated, but not yet defined, the address field is marked as "taken", i.e. set to 0. When an immediate attribute is created or a synthesized attribute defined and allocated storage space, the storage manager is called to get a pointer to free storage in the stringspace. The pointer that is returned is stored in the address field of the associated node in the nodelist. Simultaneously, the node index is stored in the index area of the stringspace. Figure 5. shows the nodelist with the three attributes contained in the inheritance stack shown in Figure 2.

```
|------|---|---|------|---|---|------|---|---|-----
| 9ED0 | 0 | 0 | 9ED8 | 0 | 0 | 9EE0 | 0 | 0 |
|      |   |   |      |   |   |      |   |   |
| ADDR |TAG|UNU|      |   |   |      |   |   |. . .
|  2b  |1b |1b |      |   |   |      |   |   |
|------|---|---|------|---|---|------|---|---|-----
|              |                  |             |
|   4b node    |                  |             |
```

FIGURE 5. NodeList With Three Attributes

2.0.3 STRINGSPACE

Data objects are stored as strings in the stringspace. Each string that represents a data object has a 5-byte header. The first two bytes contain the index (IDX) back to the corresponding node in the nodelist. The third byte contains the type (TYP) of attribute the string represents. Attribute types are discussed in paragraph 2.2, below. The last two bytes of the header contains a 2-byte relative displacement (NXT) to the next node in stringspace. NXT represents the number of bytes from IDX of the current string to IDX of the next string or free storage. Figure 6 represents how storage appears with two attributes, a symbol representing " ABC" (See Figure 6a), and a numeric attribute, # 123 (Figure 6b.)

```
|---------|----|---------|----|----|----|
|   01    | D0 |   08    | 41 | 42 | 43 |
|         |    |         |    |    |    |
|   IDX   |TYPE|   NXT   |DATA|DATA|DATA| . . .
|   2b    | 1b |   2b    | 1b | 1b | 1b |
|---------|----|---------|----|----|----|
|                        |
|    5-byte header       |
```

FIGURE 6a. First Attribute: " ABC".

```
|---------|----|---------|----|----|----|----|
|   02    | C2 |   09    | 02 | 00 | 01 | 23 |
|         |    |         |    |    |    |    |
|   IDX   |TYPE|   NXT   |WHL |FRC |DATA|DATA| . . .
|   2b    | 1b |   2b    | 1b | 1b | 1b | 1b |
|---------|----|---------|----|----|----|----|
|                        |
|    5-byte header       |
```

FIGURE 6b. Second Attribute: # 123.

In Figure 6b, WHL specifies that two bytes of packed BCD data are to be considered as whole numbers. In this example, the first byte is 01 and the second byte is 23. Together, these bytes comprise the number +123. A full explanation of the string representation of a number follows in paragraph 2.2.1.

2.2 DATA TYPES

Data typing of FORJR objects corresponds to types of data described in the BADJR Report with the exception of STREAMS. At the present time, STREAM processing is not implemented in FORJR. The following shows the types of data represented in FORJR.

OBJECT	TYPE (HEX)
NEG. NUMBER	C1
POS. NUMBER	C2
SYMBOL	D0
BOOLEAN	D0
SEQUENCE	E0

2.2.1 NUMBERS (Type C1 or C2)

In FORJR numbers, the type field indicates the sign of the number, type C1 for negative numbers, type C2 for positive numbers. FORJR stores decimal digits in packed BCD format with two decimal digits per byte. FORJR arithmetic is accomplished in decimal. Figure 6b showed a numeric string, # 123, with two additional fields, WHL (for WHOLE NUMBER), and FRC (for FRACTION.) These fields indicate the number of packed BCD bytes to the left and right, respectively, of the implied decimal point. Therefore, the first whole byte of the number may have a leading zero digit to align the bytes properly. Since WHL and FRC are 1-byte hex numbers, FORJR can represent at most 256 decimal digits to the right of the decimal point and 256 digits to the left of the decimal decimal point. These two fields are always stored, even if no digits are represented. So, a numeric string has a minimum of seven bytes, the 5-byte header, and one byte each for WHL and FRC.

2.2.2 SYMBOLS (Type D0)

Symbols are stored as lists of characters represented by ASCII values with one byte per character. Figure 6a showed the stringspace for the symbol " ABC". An N-character symbol is stored in N bytes. Therefore, NXT-5 gives the length of the symbol, so a separate length field is unnecessary. A symbolic string with no symbols is considered EMPTY.

2.2.3 BOOLEAN (Type D0)

Boolean strings are a special case of symbolic strings. In order to be classified as a boolean node, a symbol must begin with T for a TRUE value or F for a FALSE value. Any attempt to use a symbol (as a boolean value) that does not begin with T or F will generate an error.

2.2.4 SEQUENCES (Type E0)

FORJR stores sequences as lists of indices of the objects that comprise the sequence. The indices (NODEPTR, a 2-byte hex number that points to nodes in the nodelist) are stored in the stringspace of the sequence in the same order as the elements appear in the sequence. An N-element sequence has 2*N data bytes, plus the 5-byte header. A separate count field is unnecessary because (NXT-5)/2 gives the number of elements in the sequence. The NXT field of a sequence of zero elements (NIL sequence) is exactly equal to five. Figure 7 shows a sequence constructed of the numeric (NODEPTR 01) and symbolic (NODEPTR 02) strings from Figure 6.

```
|--------|----|--------|--------|--------|----
|   03   | E0 |   09   |   01   |   02   |
|        |    |        |        |        |
|  IDX   |TYPE|  NXT   |NODEPTR |NODEPTR |  . . .
|  2b    | 1b |  2b    |   2b   |   2b   |
|--------|----|--------|--------|--------|----
|                      |
|   5-byte header      |
```

FIGURE 7. Sequence of One Symbol and One Number

2.3 GARBAGE COLLECTION AND STORAGE COMPACTION

Because of the single-assignment rule in BADJR, many temporary objects are generated in the storage areas. To conserve storage space FORJR uses a node-tagging scheme to implement garbage collection.

The inheritance stack rises and falls as FORJR routines are called and results returned. Any critical attribute that needs to be used again is either in the active stack or is referred to by another attribute. Therefore, it is safe to collect any unreferenced nodes.

Garbage collection can be invoked by the user via COLECT or can be initiated by FORJR itself if free storage, either nodes or stringspace, is exausted. When collection begins, all computation is halted to ensure storage remains fixed until collection is complete.

Every node in the nodespace whose index is referenced in the active inheritance stack is tagged by setting the tag field to the current value of the marker, a value which alternates between 0 and 1. Therefore, the tag field of a node is ONLY changed if it is not to be collected. If a node is a sequence, its elements are marked recursively until all referenced elements are marked.

After tagging, all nodes in the nodelist are checked for the current tag value. Any node with the incorrect value has its address field set to FFFFH indicating this is a collectable node. The IDX field of the corresponding stringspace is also set to collectable.

After all nodes and stringspaces have been checked, storage is compacted using a common method. Starting at the base of the stringspace area, the compactor checks the IDX field of each stringspace to see if it has been marked for collection. If a stringspace is collectable, successive stringspaces are examined until the first uncollectable stringspace is encountered. The uncollected stringspace is then "slid up" to the address of the first collectable stringspace. This check-and-slide process is repeated until all uncollectable stringspaces are adjacent with no holes between them or the beginning of free space is encountered. As uncollected stringspaces are moved, the corresponding pointers in the nodelist are updated to reflect the new address of the stringspace.

If a user invokes the garbage collector via COLECT, and no collectable space exists, the message:

NO GARBAGE FOUND

is printed on the console.

IV. FORJR INSTRUCTION SYNTAX

1.0 INTRODUCTION

FORJR provides three levels of instruction, IMMEDIATE, PRIMITIVE, and RELATIONAL. Simple examples explaining the use of FORJR instructions are provided below. The actual FORTH definitions of the FORJR Syntax can be found in Appendix B, Forth Screen Contents.

The following is a list of FORTH words and their respective functions. The definition of these functions are provided to assist in understanding the FORJR INSTRUCTION SYNTAX:

INITSTORE - Initializes the INHERITANCE STACK, NODELIST, and the STRINGSPACE storage areas.

{ - Starts a new FRAME on the FRAME stack. (A more complete description is contained in Para. 4.12.5)

} - Symbolizes the end of the FRAME construction but is for readibility only.

DEFLOC - Provides a synthesized attribute to be defined later by some FORJR routine.

A1 A2 A3 ... A11 - Stacks attributes 1 2 3 ... 11 respectively from the previous FRAME onto the current FRAME. (The same operation can be accomplished by 1 STKINH 2 STKINH etc.)

2.0 IMMEDIATE INSTRUCTIONS

Immediate instructions produce a single attribute with a specified attribute type. FORJR defines the following immediate functions:

NUMERIC CONSTANT

SYMBOLIC CONSTANT

SELECT FUNCTION

LENGTH FUNCTION

CONSTRUCT FUNCTION

MERGE FUNCTION

SPECIAL NOTES:

(1) Data input integrity is extremely critical for the IMMEDIATE INSTRUCTIONS. Recovery from mistyped entries may cause FORJR to abort, particularly when using immediate number or symbol builders inside SEQUENCES.

(2) Prior to executing ANY FORJR instructions, the data storage areas must be initialized via INITSTORE.

2.1 NUMERIC CONSTANTS (# ...)

An immediate number attribute may be created with a maximum of 256 digits, including sign and decimal point. Negative numbers must be preceded by a - sign. However, the + sign is optional for positive numbers. The input string may contain at most one (1) decimal point and no imbedded blanks. The pound sign (#) followed by one or more blanks invokes the immediate numeric constant function. A blank or carriage return following the desired number terminates the immediate constant function.

EXAMPLE: # 123 (Creates a positive attribute)

EXAMPLE: # -456.789 (Creates a negative attribute)

In the run time environment, immediate number constants may also be created using the RDNUM function. After invoking RDNUM, you may enter the desired sign, number, and decimal point followed by a carriage return.

2.2 SYMBOLIC CONSTANTS (" aaa")

An immediate symbolic constant can be created by bracketing a character string in double quotes (" aaa"). A maximum of 256 ASCII symbols may be contained in the input character string. All printable ASCII characters (except double quote (") and control characters) may be included in the symbol. The symbolic constant builder is invoked with a double quote (") followed by one (1) blank. The desired character string can be terminated with either an ending double quote (") or a carriage return. Any symbolic attribute created inside a FORTH definition must terminate with the quotes. As with numeric constants, a runtime facility, RDSYM, exists to read in characters from the keyboard. In this case, a carriage return terminates the symbol construction.

EXAMPLE: " ABC"

EXAMPLE: " ABC<cr> (symbol is same as above)

2.3 SELECT (SL or SR)

The SELECT function creates an attribute from selected element of a target SEQUENCE. Both SELECTRIGHT (SR) and SELECTLEFT (SL) are available in FORJR. SL choses an element indexed into the sequence from the left, while SR choses the elements indexed from the right. The target sequence or a copy of the target sequence must be the top-most attribute in the frame, otherwise an error will occur. In addition, the index value must be less than or equal to the length of the sequence. To execute the SELECT function, the index value of the desired element is put onto the FORTH stack. After SL or SR is executed, the sequence on top of the FRAME stack will be replaced by the desired element from the sequence.

(In the following examples assume the target SEQUENCE is on top of the inheritance stack and contains 4 elements.)

EXAMPLE: 1 SL (Replaces the top sequence with the first element of the sequence.)

EXAMPLE: 4 SR (Also will replace the top sequence with the first element of the sequence.)

2.4 LENGTH (LENGTH)

The LENGTH function creates a numeric attribute representing the number of elements in a target sequence. The target sequence must be the topmost attribute on the current frame stack.

EXAMPLE: LENGTH (Replaces the top sequence attribute with a numeric attribute containing the number of elements in the sequence.)

2.5 CONSTRUCT (<< . . . >>)

The CONSTRUCT function combines one or more attributes into a single sequence. Other immediate instructions can be nested inside the construct operator. A pair of adjacent "less than" symbols, <<, invokes the CONSTRUCTOR while a pair of "greater than" symbols, >>, terminates the CONSTRUCTOR. The desired elements are contained between << and >>. The sequence constructor can be nested to provide sequences within sequences.

EXAMPLE: { << # 1 " test" >> } (Creates a two-element sequence containing one numeric and one symbolic element.)

EXAMPLE: { << A1 A2 A3 >> } (This example assumes 3 attributes are in the current frame. A new frame is created and a sequence attribute is constructed from three attributes from the original frame.)

EXAMPLE: { << RDSYM >> } (Makes a 1-element sequence from characters input from the keyboard. The element is a symbol representing the input string.)

EXAMPLE: { << # 1 << # 2 " ABC" >> >> } (Creates a two-element sequence. The first element is an immediate number, the second element is a two-element sequence of an immediate number and immediate symbol.)

2.6 MERGE (MERGE . . . CLSMER)

The MERGE function operates on one or more sequences and produces a single sequence containing all the elements from the enclosed sequences.

EXAMPLE: MERGE A1 A2 CLSMER (Makes a sequence of the elements of both attribute 1 and attribute 2 of the current frame. NOTE: both attributes must be sequences.)

EXAMPLE: MERGE << # 1 >> << " THIS IS A TEST" >> CLSMER (Creates a sequence of two elements, an immediate numeric element and an immediate symbolic element.)

3.0 PRIMITIVE INSTRUCTIONS

Each primitive instruction has a predetermined number of inherited and synthesized attributes. The number of inherited attributes varies depending upon the type of instruction. Only one synthesized attribute is defined by a primitive function.

FORJR handles the following types of primitive instructions:

CHARACTERISTIC FUNCTIONS

CONVERSIONS

SEQUENCE MANIPULATIONS

ARITHMETIC OPERATORS

3.1 CHARACTERISTIC FUNCTIONS

Characteristic functions are designed to test the type of an inherited attribute. These functions use one inherited attribute as input, which can be any object, and synthesizes one boolean attribute. The boolean attribute will have the value of T (for TRUE) or F (for FALSE) depending upon the results of the test. FORJR characteristic functions include: ATOM?, NIL?, SYMBOL?, NUMBER?, BOOLEAN?, EMPTY?, and SEQUENCE? Most of these functions just examine the type field of the inherited attribute and define the boolean attribute accordingly. The two functions NIL? (for sequences) and EMPTY? (for symbols) return T if the number of data bytes in the stringspace of the inherited attribute is zero, and F otherwise. In addition, F will be returned if NIL? is applied to a NON-sequence or EMPTY? is applied to a NON-symbol.

(In the following examples, assume that a frame exists containing an inherited attribute and an undefined synthesized attribute.)

EXAMPLE: { A1 A2 } NUMBER? (Starts a new frame and stacks an inherited attribute (A1) and a synthesized attribute, (A2). The type field of the first attribute is checked and defines A2 as a boolean T if A1 is a numeric attribute, F otherwise. The frame is then reset back to the original frame.)

3.2 CONVERSIONS

FORJR has no automatic or default conversions. Therefore, any conversion must be accomplished through explicit conversion functions. These functions use one inherited and one synthesized attribute. The names of most of the conversion functions identify the type of conversion being accomplished. The first three letters of the function name indicate the type of the inherited attribute and the last three letters indicate the desired conversion. Type checking is performed on the inherited attribute. Therefore, if the type does not match the desired input, an error message is printed and the conversion is aborted. The cnly exception to the naming convention is the IDENTITY function, which makes a duplicate of any inherited attribute.

FORJR provides the following conversions:

SYMBOL-TO-SEQUENCE

SEQUENCE-TO-SYMBOL

SEQUENCE-TO-NUMBER

NUMBER-TO-SYMBOL

IDENTITY

3.2.1 SYMBOL-TO-SEQUENCE (SYMSEQ)

SYMSEQ creates a new symbol in the stringspace for each ASCII character in the inherited attribute. The synthesized attribute becomes a sequence of the new symbol nodes.

```
EXAMPLE:  { " ABCD"  DEFLOC }
          { A1 A2 } SYMSEQ
```

(Using the symbol ABCD from the first frame, a sequence attribute with four elements, A, B, C, D, is defined in the second attribute.)

3.2.2 SEQUENCE-TO-SYMBOL (SEQSYM)

SEQSYM creates a new symbol containing the elements of the sequence. If the sequence contains any NON-symbolic elements, an error message is printed and the conversion is aborted.

```
EXAMPLE:   { << " AB"  " CD" >> DEFLOC }
```

(The first attribute is a two element sequence)

```
          { A1 A2 } SEQSYM
```

(A symbolic attribute, ABCD, is created in the second attribute.)

3.2.3 SEQUENCE-TO-NUMBER (SEQNUM)

SEQNUM operates on a sequence whose elements are symbols representing the digits 0-9, + or -, and at most one decimal point. SEQNUM will convert the sequence into a numeric attribute whose digits match the elements of the sequence. The elements may be a series of symbols, or a single character string.

```
EXAMPLE:   { << " -12.34" >> DEFLOC }
```

(Creates a sequence with six symbolic elements, -, 1, 2, 3, ., and 4. DEFLOC provides a synthesized attribute.)

```
          { A1 A2 } SEQNUM
```

(Creates a numeric attribute, -12.34 in the second attribute.)

```
EXAMPLE: { << " -" " 1" " 2" " ." " 3" " 4" >> DEFLOC }
          { A1 A2 } SEQNUM
```

(Has the same effect as the above example.)

3.2.4 NUMBER-TO-SYMBOL (NUMSYM)

NUMSYM creates a symbolic attribute which represents the sign, decimal point, and digits of a number.

EXAMPLE: { # 123 DEFLOC }

(Creates a one numeric and one synthesized attribute.)

{ A1 A2 } NUMSYM

(Generates a symbolic attribute that is the ASCII representation of the number +123.)

3.2.5 IDENTITY (ID)

The IDENTITY function creates an exact duplicate of any defined object, including numbers, symbols, and sequences.

EXAMPLE: { " test" DEFLOC }

(A symbol, test, is created and a synthesized attribute provided.)

{ A1 A2 } ID

(Makes the second attribute an exact duplicate of the first.)

3.3 SEQUENCE MANIPULATIONS

Major order and space transformations are performed on sequences in FORJR. These manipulation functions consist of:

DISTRIBUTION

REVERSE

SELECTION

3.3.1 DISTRIBUTION (DL or DR)

There are two forms of the distribution function, DL (DISTRIBUTE-LEFT) and DR (DISTRIBUTE-RIGHT). Each version must have two inherited attributes and one synthesized attribute. The first inherited attribute must be a sequence, the other some object. After the function call, the synthesized attribute becomes a sequence with the same length as the inherited sequence. Each element of the new sequence is a sequence of length two consisting of an individual elements from the original sequence prefixed (DL) or suffixed (DR) with the object.

EXAMPLE: { << # 34 # 56 >> " ABC" DEFLOC }

(A frame with three attributes, (1) a 2-element sequence, (2) the symbol ABC, (3) a synthesized attribute.)

{ A1 A2 A3 } DL

(Defines the third attribute as a sequence with the following characteristics:

<< << " ABC" # 34 >> << " ABC" # 56 >> >> .)

3.3.2 REVERSE (RV)

The REVERSE function makes a sequence by copying all the elements of the inherited sequence in reverse order.

EXAMPLE: { << # 1 # 2 # 3 >> DEFLOC }

(A frame with two attributes, (1) a 3-element sequence, (2) a synthesized attribute.)

{ A1 A2 } RV

(Defines the synthesized attribute as a 3 element sequence:

<< # 3 # 2 # 1 >>   .)

3.3.3 SELECT (SEL or SER)

The primitive SELECT is not to be confused with the immediate SELECT function. The primitive SELECT operates entirely from attributes, including the index of the desired sequence element. The number represented by the numeric attribute must be equal to or less than the length of the sequence. After the SELECT function call, the synthesized attribute is defined as the selected element of the sequence.

EXAMPLE: { << # 123 # 456 # 789 >> # 2 DEFLOC }

(A frame with three attributes, (1) a 3-element sequence, (2) a numeric attribute, (3) a synthesized attribute.)

{ A1 A2 A3 } SEL

(Defines the synthesized attribute with the second element of the sequence, i.e. the number +456.)

3.4 ARITHMETIC OPERATORS

FORJR numbers are implemented as fixed point decimals and stored in packed BCD format. All attributes used as operands should be numeric types. After computation, the result is normalized before storing in the stringspace. Normalization is accomplished by stripping leading or trailing zeros. However, because the decimal point falls on a byte boundary, there may be one leading zero digit and trailing zero digit.

FORJR provides the following arithmetic functions:

ADDITION

SUBTRACTION

MULTIPLICATION

DIVISION

ABSOLUTE VALUE

NEGATION

INTEGER

3.4.1 ADDITION (AD)

For addition and subtraction the number of digits to the right of the decimal point in the result is the same as the larger of the two operands. The addition operator uses two numeric attributes and defines a synthesized attribute as the sum of the two numbers.

EXAMPLE: { # 1 # 2 DEFLOC }

(Establish a frame with two numeric and one synthesized attribute.)

{ A1 A2 A3 } AD

(The synthesized attribute is defined and represents the number +3.)

3.4.2 SUBTRACTION (SB)

This operator uses two numeric attributes and defines a synthesized attribute as the difference of the two numbers.

EXAMPLE: { # 10 # 15 DEFLOC }

(Establish a frame with two numeric and one synthesized attributes.)

{ A1 A2 A3 } SB

(The synthesized attribute is defined as -5.)

3.4.3 MULTIPLICATION (ML)

In multiplication, the number of significant digits in the result is computed as the sum of significant digits in the operands, normalized as above. The multiplication operator uses two numeric attributes and defines a synthesized attribute as the product of the two numbers.

EXAMPLE: { # 2 # 6 DEFLOC }

(Establish a frame with two numeric and one synthesized attribute.)

{ A1 A2 A3 } ML

(The synthesized attribute is defined as +12.)

3.4.4 DIVIDE (DV)

The divide operator will always produce at least six decimal digits normalized as above. This operator uses two numeric attributes and defines a synthesized attribute as the dividend of the two. Division by zero is prohibited. If an attempt is made to divide by zero, the operation will be aborted, and the synthesized attribute will remain undefined.

EXAMPLE: { # -2.3 # 2 DEFLOC }

(Establish a frame with two numeric and one synthesized attribute.)

{ A1 A2 A3 } DV

(The synthesized attribute is defined as -1.15.)

EXAMPLE: { # 1 # 0 DEFLOC }

(Establish a frame with two numeric and one synthesized attributes.)

{ A1 A2 A3 } DV

(An error is generated because of the attempt at division by zero. The synthesized attribute remains undefined.)

3.4.5 ABSOLUTE VALUE (AB)

This operator makes a copy of the inherited numeric attribute but sets the type field to a positive numeric value.

EXAMPLE: { # -1.23 DEFLOC }

(Establish a frame with a negative numeric attribute and a synthesized attribute.)

{ A1 A2 } AB

(Defines the synthesized attribute as +1.23.)

3.4.6 NEGATION (NG)

This operator produces a copy of the inherited numeric attribute but changes the sign of the number by reversing the type field to the opposite of the original number.

EXAMPLE: { # +4.56 DEFLOC }

(Establish a frame with a positive numeric attribute and a synthesized attribute.)

{ A1 A2 } NG

(Defines the synthesized attribute as -4.56.)

3.4.7 INTEGER (INT)

This operator defines a synthesized attribute with just the integer portion of an inherited numeric attribute.

EXAMPLE: { #16.789 DEFLOC }

(Establish a frame with a numeric attribute representing the number 16.789 and a synthesized attribute.)

{ A1 A2 } INT

(Defines the synthesized attribute as +16.)

3.4.8 MOD (MD)

The MOD function operates in standard manner, producing only the remainder as an integer . The function uses two numeric attributes and defines a synthesized attribute as the MOD of the two numbers. The input numbers are first converted to integers via the INT function described above.

EXAMPLE: { # 180 # 25 DEFLOC }

(Establish a frame with two numeric and one synthesized attribute.)

{ A1 A2 A3 } MD

(The synthesized attribute is defined as +5.)

(180 MOD 25 = 5.)

3.5 LOGICAL OPERATORS

The normal logical operations AND, OR, Exclusive OR, and NOT are provided in FORJR. The FORJR names for these function calls are: BAND, BOR, BXOR, and BNOT, respectively. With the exception of BNOT, each operates on two inherited boolean attributes and defines a synthesized attribute with the appropriate boolean value, TRUE (T), or FALSE (F). BNOT uses only one inherited and one synthesized attribute.

(For each of the following examples, use the frame:

{ " T" " F" " T" DEFLOC }

Where the first three attributes are boolean attributes and the forth is a synthesized attribute.)

EXAMPLE: { A1 A3 A4 } BAND

(Defines the synthesized attribute as a boolean TRUE.)

EXAMPLE: { A1 A2 A4 } BOR

(Defines the synthesized attribute as a boolean TRUE.)

EXAMPLE: { A1 A3 A4 } BXOR

(Defines the synthesized attribute as a boolean FALSE.)

EXAMPLE: { A1 A4 } BNOT

(Defines the synthesized attribute as a boolean FALSE.)

3.6 RELATIONAL OPERATORS

The relational operators discussed in the BADJR report compare the types and values of two inherited attributes. The precedence order used for comparing attributes is as follows:

NUMBERS < SYMBOLS < SEQUENCES.

A synthesized attribute is defined with a boolean value, TRUE (T) or FALSE (F) as a result of the comparison. If the attributes in the comparison are sequences, the relational operators check the sequence lengths and considers shorter sequence as preceding longer sequences. If the sequences are of the same length, the relational operator compares the individual elements inside the sequences and awards precedence based on the above criteria and defines the synthesized attribute accordingly. The FORJR names for the relational instructions are:

EQ?

NE?

LT?

LE?

GT?

GE?

EXAMPLE: { # 3.1 # 2.5 DEFLOC }

(Establish a frame with two numeric and one synthesized attribute.)

{ A1 A2 A3 } GT?

(Since 3.1 is greater than 2.5, the synthesized attribute is defines as TRUE (T).)

EXAMPLE: { " ABC" # 123 DEFLOC }

(Establish a frame with one symbolic atom, one numeric atom, and a synthesized attribute.)

{ A1 A2 A3 } LE?

(Because of the precedence order established between symbols and numbers, i.e. NUMBERS < SYMBOLS, the synthesized attribute is defined as FALSE (F).)

EXAMPLE: { # 999 << # 0 >> DEFLOC }

(Establish a frame with one numeric atom, a sequence containing one numeric atom, and a synthesized attribute.)

```
{ A1 A2 A3 } GT?
```

(Because atoms have a lower precedence value than sequences, the synthesized attribute would be defined as a boolean FALSE (F).)

EXAMPLE: `{ << " A" >> << # 1 # 2 # 3 >> DEFLOC }`

(Establish a frame with two sequences and one synthsized attribute. The the first sequence contains one symbolic atom the second sequence contains three numeric atoms)

```
{ A1 A2 A3 } LT?
```

(The synthesized attribute is defined as TRUE (T) because the length of the first sequence is one as compared to a length of three for the second sequence.)

EXAMPLE: `{ << " CAT" >> << " DOG" >> DEFLOC }`

(Establish a frame with two sequences and one synthesized attribute.)

```
{ A1 A2 A3 } GT?
```

(Since the sequence lengths are equal, the relational instruction must compare the contents of each sequence. Since CAT is NOT lexigraphically "greater than" DOG, the synthesized attribute is FALSE (F).)

4.0 OTHER FORJR INSTRUCTIONS

Along with the FORJR instructions listed in the introduction to Section IV, there are a number of FORJR instructions dealing with the FORJR environment. Examples are included if the function call involves frame manipulation.

4.1 I/O FUNCTIONS (RDNUM, RDSYM, PRNUM, PRSYM, PRBUL)

To prevent conflicting file handling problems all I/O operations are done from FORTH. Number and character input routines (RDNUM, RDSYM) are immediate and described in paragraph 2.0, above. However, output functions (PRNUM, PRSYM, PRBUL) act as primitive operators and must function on inherited attributes.

EXAMPLE: { # 1.23 " TRUE" }

(Establish a frame with a numeric and symbolic attribute. Use this frame for the following examples.)

```
{ A1 } PRNUM
```

(Results in a console output: +1.23).

```
{ A2 } PRSYM
```

(Results in a console output: TRUE)

```
{ A2 } PRBUL
```

Since the symbolic attribute begins with a "T", the boolean print operator will also function on this attribute. If the boolean print operator is applied to a NON-boolean attribute, an error occurs.

EXAMPLE: { A2 } PRBUL

(Results in a console output:

BOOLEAN VALUE = TRUE.)

4.2 FRAME STATUS (FRAME)

The FORJR word FRAME causes a dump of the current frame providing the beginning address of the current frame on the INHERITANCE stack. The type of each attribute in the frame is printed, and if the attribute is a number or symbol, the attribute itself is printed. However, if the attribute is a sequence, only the sequence length is printed.

4.3 MEMORY STATUS (DUMPINH, DUMPNOD, DUMPSTR)

The memory status words execute 256-byte dumps of the respective memory areas, the FRAME STACK, the NODESPACE, and the STRINGSPACE.

4.4 POP ATTRIBUTE (POPINH)

The word POPINH deletes the top attribute from the current frame.

4.5 RESET FRAME (RSTINH)

RSTINH resets the frame back to the original (previous) frame.

4.6 GARBAGE COLLECTOR (COLECT)

A full description of the garbage collection system is provided in Section II, paragraph 4.3, GARBAGE COLLECTION AND STORAGE COMPACTION.

4.7 EXECUTION CONTROL (QUES)

The function QUES interrogates a boolean attribute and returns a one (1) to the FORTH stack if the boolean is TRUE (T), or a zero (0) if the boolean is FALSE (F). Flow of execution through a FORJR line is accomplished using standard FORTH if-then-else convention.

EXAMPLE: (Write a FORTH test routine that prints the larger of two numbers from a frame.)

```
{ # 123 # 456 DEFLOC }
```

(Establish a frame with two numeric attributes, and one synthesized attribute.)

```
: TEST&PRINT ( FORTH test routine )
{ A1 A2 A3 } GT?   (IS A1 > A2 ? )
{ A3 } QUES    (Test the boolean attribute)
IF { A1 } PRNUM
ELSE { A2 } PRNUM ENDIF ;
```

4.8 FRAME SLIDER (SLIDE)

The FORJR function SLIDE moves the current frame down on top of the previous frame. This is designed to optimize utilization of memory space and facilitates recursive FORJR calls.

4.9 ENHANCED FORJR SYNTAX

Several FORJR words have have been defined that make FORJR syntax resemble more closely the BADJR syntax as given in the original BADJR Report. The same functions are provided in other forms, but these words simplify programming in FORJR and provide more readable code. Some of the enhanced syntax functions can be used in a "live" environment, while others are designed to be used inside FORJR function definitions.

4.9.1 ATTRIBUTE NAMING/STACKING CONVENTIONS

A FORJR compile time facility allows the user to refer to attributes by name rather than by number. Because each attribute name is given a separate FORTH dictionary entry, it is not advisable to put this facility inside a FORJR program definition. In order to use this facility, the user must follow the syntax precisely.

EXAMPLE: {{ ^ xxx ^ yyy ^^ aaa ^^ bbb }}

The attribute naming procedure is initiated by a pair of adjacent left "curly brackets", {{. There must be no spaces between the two left brackets. A right pair has been provided but is for readability only.

A single "up carat" followed by some character string associates an integer value with the attribute stacking routine, STKINH. The variable ATTCOUNT is initialized to zero via {{. Every time ^ or ^^ is used, ATTCOUNT is incremented by one. The new value of ATTCOUNT is included in the definition of the current attribute being named. In the above example, xxx becomes a FORTH word with the following characteristics:

```
: xxx 1 STKINH ;
```

When executed, xxx stacks the first attribute from the previous frame onto the current frame. The FORTH word yyy would stack the second attribute.

The double carat, ^^ , assigns the next integer in ATTCOUNT to a character string and causes the string to behave as the single carat routine. However, the double carat routine implies that the attribute referenced is a local attribute. A variable, LOCCOUNT, keeps track of the total number of local attributes desired. In the example, aaa has a definition resembling:

```
    : aaa 3 STKINH
```

The function bbb is defined as:

```
    : bbb 4 STKINH ;
```

4.9.2 DEFINING SYNTHESIZED ATTRIBUTES (LOC)

Using the value contained in LOCCOUNT as described above, the desired number of local attributes can be requested quickly and easily via the function LOC. A loop is performed that executes the function DEFLOC once for each local attribute desired. In the above example, ^^ is used twice, LOCCOUNT is two, and two local attributes would be created. The function aaa would stack the first local attribute, bbb the second.

The LOC facility has a limitation that dictates it MUST be used inside a FORJR program definition. Any attempt to use LOC in a live environment will produce nil results.

4.9.3 INHERITED¦SYNTHESIZED ATTRIBUTE SEPARATOR (¦)

The dummy FORJR command, ¦ ,exists that enhances readability. This function does nothing, but when used it becomes readily apparent which attributes are inherited, and which are synthesized.

4.10 TOPMOST ATTRIBUTE STACKER (>**)

Occasionally, an attribute is generated on top of the current frame but the user does not know which attribute number it is. Although FRAME lists out all the attributes in the current frame, executing FRAME inside a program definition may not be desirable. Therefore, a facility exists, >**, that stacks the topmost attribute from the previous frame onto the current frame.

4.11 SEQUENCE LENGTH (SEQLEN)

This function returns to the FORTH stack an integer value that is the length (number of elements) of a sequence. The desired sequence must be the topmost attribute, or the only attribute in a frame because after SEQLEN is called, the frame is reset back to the previous frame. The best way to use this facility is to start a new frame and stack the desired sequence onto it and then call SEQLEN.

```
EXAMPLE:  { << # 1 # 22 # 33 >> }
```

(Establish a frame with a sequence of three elements.)

```
          { A1 }  SEQLEN
```

(Results in the number 3 on the FORTH stack.)

4.12 FORJR RECURSIVE INSTRUCTIONS

Certain FORTH instructions provide recursive capabilities for FORJR lines. These instructions themselves do not interface with the Z-80 assembly code but provide the environment for recursion in FORJR.

The flow of execution in FORTH is governed by the addresses of functions that are contained on the FORTH return address stack. When one FORTH word calls another FORTH word, the Program Field Address (PFA) of the next word to be executed in the calling word is pushed onto the FORTH return address stack. The principle of recursion used in FORJR is to replace this PFA on the return address stack with the PFA of the recursive routine. Every time this replacement action takes place, the recursive routine is executed again. If the recursive routine is not to be executed again, the PFA of a dummy routine is pushed onto the return address stack and execution resumes in the calling word.

4.12.1 Null FORTH Word (DUMWORD)

DUMWORD is a null FORTH routine whose address is used in the function BOL, described below.

4.12.2 FORTH Word Address Holder (EXWORD)

EXWORD is a variable used to hold the addresses of FORTH routines. The contents of this variable are put onto the FORTH return stack via EXX, described below.

4.12.3 Begining of Line Word (BOL)

BOL signifies the begining of a FORJR line. This function stores the Program Field Address (PFA) of the null routine, DUMWORD, into EXWORD.

4.12.4 Execution Address Stacker (EXX)

EXX has two functions: (1) Pushes the value of EXWORD (which is always a PFA of some FORTH word, either a dummy function, or a recursive routine); (2) Stores the PFA of

DUMWORD into EXWORD. After EXX has executed, the FORTH word whose PFA was pushed onto the return address stack is executed.

4.12.5 New Frame Starter ({)

The new frame starter, { , has two functions: (1) Executes EXX, thereby pushing the PFA contained in EXWORD onto the FORTH return address stack; (2) Starts a new frame by calling SETINH.

4.12.6 End of Line (EOL)

The end of a FORJR line is signified by EOL. This function has three responsibilities: (1) Drops the PFA of the next FORTH word to be executed from the return address stack, thereby preventing that word from executing; (2) Slides the current frame down over the preceding frame via SLIDE; (3) Calls EXX. Basically, besides calling SLIDE, EOL switches the PFA of the next word on the return address stack with the PFA contained in EXWORD.

4.12.7 Initial Function Name Setup (BADJR)

Since a dictionary entry must previously exist for every FORTH word executed, BADJR is used to create a dummy entry. BADJR, using run time procedures, defines a function with the following characteristics: (1) The function contains a variable, initially zero; (2) The function stores the value of its variable into EXWORD. The intent behind BADJR is to replace the zero in the variable with the PFA of a recursive FORJR line. Therefore, when the function is called, it sets up recursion by puttine its own PFA into

EXWORD.

EXAMPLE: BADJR FACT

4.12.8 PFA Swapping Routine (DEFINE)

DEFINE replaces the zero in the variable associated with a function set up by BADJR with the PFA of a recursive FORJR line. The calling sequence for DEFINE is:

```
[ ' function-name DEFINE ]
```

where function-name is the name of a function previously set up by BADJR. This series of commands must be contained inside the definition of a FORJR line. The square brackets, [. . .], suspend compilation of the line to perform the instructions within. [' function-name DEFINE] replaces the zero in the variable associated with function-name with the PFA of the line currently being defined. Therefore, when the routine function-name is called, the PFA of the FORJR line is stored into EXWORD.

```
EXAMPLE:     : LINE1
                   [ ' FACT DEFINE ]
                   (FORJR instructions go here). . . ;
```

Any references to FACT inside the definition of LINE1 will cause LINE1 to be executed.

5.0 FORJR RECURSIVE EXAMPLE

The following is an example of a FORJR recursive routine that computes the factorial of an input value. This example uses the enhanced FORJR syntax and recursive instructions. The Roman numerals out to the right refer to comments provided below. The function begins with a call to FACTOR which is listed in line (xiv).

EXAMPLE:

```
    {{  ^  X  ^  Y  ^  Z  ^^  A  ^^  B  ^^  C  }}   (i)
    BADJR  FACT                                    (ii)
    : LINE1                                        (iii)
        [  '  FACT  DEFINE  ]                      (iv)
        LOC                                        (v)
        {  Y  # 1  |  A  }  LE?                    (vi)
        {  A  }  QUES                              (vii)
        IF  {  # 1  X  |  Z  }  ML                 (viii)
        ELSE                                       (ix)
           {  Y  # 1  |  B  }  SB                  (x)
           {  X    Y  |  C  }  ML                  (xi)
           {  C    B  |  Z  }  FACT  ENDIF         (xii)
        EOL  ;                                     (xiii)

: FACTOR  INITSTORE                                (xiv)
    {  DEFLOC  }                                   (xv)
    {  # 1  RDNUM  |  A1  }  FACT                  (xvi)
    {  A1  }  PRNUM  EOL  ;                        (xvii)
```

COMMENTS:

(i) Provides five attribute names and associates each name with the attribute stacking routine, STKINH. In addition, sets LOCCOUNT to three thereby providing for three local attributes when LOC is executed.

(ii) Provides a dictionary entry for FACT. When FACT is called, a value associated with FACT (a PFA) is stored into EXWORD.

(iii) : LINE1 ... starts the FORJR function definition.

(iv) Assigns the PFA of LINE1 to FACT. When FACT is called now, the PFA of LINE1 is stored into EXWORD.

(v) The up-carat, ^^, is used three times in defining the attribute names. Therefore, LOC provides three local attributes.

(vi) Compares Y with an immediate numeric 1. The attribute A will be defined as a boolean T or F depending upon the results of the comparison.

(vii) Checks the boolean value of A and returns 1 or 0 to the FORTH stack if A is T or F, respectively.

(viii) Using the FORTH IF-THEN-ELSE structure, LINE1 either executes line viii or proceeds with lines ix through xii depending upon the results of lne vii.

(xii) If the ELSE condition is executed, FACT stores the PFA of LINE1 into EXWORD.

(xiii) At the end of LINE1, EOL replaces the address on top of the FORTH return address stack with the contents of EXWORD. If EXWORD contains the PFA for LINE1, LINE1 will be executed again. If EXWORD contains the PFA for DUMWORD, recursion ends and processing continues in the calling word, FACTOR.

(xiv) Sets up the dictionary entry for FACTOR and initializes the data storage areas via INITSTORE.

(xv) Defines a local attribute that will contain the factorial of the input number.

(xvi) Sets up a frame with an immediate numeric 1, an input value that is read from the keyboard via RDNUM, and the local attribute provided in line (xv). FACT puts the PFA of LINE1 into EXWORD. LINE1 is not actually executed at this time, however.

(xvii) The first { in this line causes the PFA contained in EXWORD to be pushed onto the FORTH return stack which in this case is the PFA for LINE1. After the return from LINE1, the result is printed via PRNUM. Another EOL is executed sliding the current frame down over the previous frame.

CONCLUSION

The primary objective to implement an interactive BADJR functional programming machine was achieved by the FORJR project. The only BADJR functions currently not implemented in FORJR are STREAM processing and the higher level functions as contained in the BADJR Report. The structure of FORJR dictionary entries provided a syntax that closely resembled BADJR. Because FORJR is interactive, it was more difficult to compare the processing speed of FORJR versus other implementations of BADJR. Outward appearances suggest FORJR is rather slow. However, its interactive behavior may compensate for its speed.

FORJR can be run on systems with CP/M based operating systems. A limiting factor might be its size. Currently, FORJR requires over 53k of storage to load and execute, and only 8k of FORTH User Dictionary space is available.

Programming in FORJR should be relatively easy for those individuals already familiar with FORTH. Frame building, attribute passing, and the effects on storage after FORJR function calls are areas of FORJR one should become most familiar with first. After achieving a thorough understanding of these aspects of FORJR, experimenting with recursive FORJR functions can be examined. The interactive behavior of FORJR allows simple FORJR functions to be built and tested in a live environment. However, more complex functions should be created in FORTH screens to be loaded and tested. As one studies the workings of FORJR, extensive use of the frame print and storage area dump routines is suggested. Through the use of these facilities, the user can see the effects that FORJR commands have on the different storage areas and how these areas are related.

Future extensions to FORJR might involve implementation of some of the high level BADJR functions. Since the addresses of all areas of the data structures are available in FORJR, implementing the high level functions that involve sequences seems plausible. Another consideration is modifying the size of FORJR. Developing a paging scheme that swaps out the unused portions of the Z-80 assembly code is another possible area of investigation.

An interesting observation was made while developing the FORJR system. The successful linking of FORTH to another separate and distinct system seems to suggest that FORTH can be appended to the front of other systems, thereby extending and providing increased flexibility to these systems as well.

APPENDIX A

SYSTEMS PROGRAMMER GUIDE

1. USING FORJR

The FORJR system combines a FORTH full-screen editor system with Z-80 assembly language modules which have been merged into a single executable file, FORJR.COM. Normally, FORJR can be run under CP/M simply by typing:

```
FORJR
```

However, the loader in some systems is over written when the FORJR system is invoked. In these cases, FORJR can be loaded and executed using CP/M's Dynamic Debugging Tool (DDT). The format for this method is:

```
DDT FORJR.COM
```

DDT will load the FORJR system beginning at address 100h. After the load is complete, type:

```
G100
```

If loading under DDT, the system will not come up with a valid .SCR file. You must specify any desired screen file via the USING command:

```
USING filename
```

Where "filename" is the name of the desired screen file. (NOTE: The desired file MUST have a .SCR extension.)

Prior to executing ANY FORJR commands, it is IMPERATIVE that the data storage areas be initialized via:

INITSTORE

If you fail to do this, FORJR loses track of itself and the system will have to be reset. If you define test programs inside FORTH words, it is suggested that you include INITSTORE as part of the function definition.

2. MODIFYING Z-80 SOURCE FILES

If desired, the Z-80 source modules of FORJR can be modified to expand the scope of FORJR. Also, smaller versions of FORJR can be created by deleting unnecessary modules.

There are thirteen separate Z-80 assembly language source files that are used in FORJR. Table C-1 is a list of these source files with a short description of the functions of each module. In addition, the major subroutines of each module are listed. However, the user does not have ready access to all the subroutines listed. All necessary FORJR files are available on one 8" CP/M floppy disk.

The files MACROS.MAC and EQATMO.MAC do not generate any Z-80 code themselves. MACROS.MAC contains the macros used in the original ZADJR system. This file gives the user an idea of the original syntax for ZBADJR and what parameters each module anticipated. MACROS.MAC is not used in FORJR and is provided for informational purposes only.

EQATMO.MAC contains constant definitions that are used throughout the Z-80 code. The values defined are available via the M80 'EQU' pseudo-op. EQATMO.MAC must be present if any Z-80 modules are modified and reassembled.

The Z-80 files can be modified using the CP/M editor function, ED. At the begining of each file is a list of changes made including the date the change was applied. In addition, the comment field of each change also contains the date the change was applied. It is suggested that as you make changes to the code, these dating procedures be adhered to and updated accordingly.

After the desired changes have been applied to the module, it must be reassembled. Certain switches are used for assembling the Z-80 modules. The command used to assemble the Z-80 code is:

```
M80 ,=filename/L/M/R/Z
```

Where:

L = Forces generation of a listing file, filename.PRN.

M = Initializes block data areas to zero.

R = Forces generation of an object file, filename.REL.

Z = Assembles Z-80 opcodes.

Each of the created files, .PRN and .REL will have the same filename as the .MAC file.

3. LINKING Z-80 FILES INTO THE FORJR SYSTEM

Because FORJR must know the location of the Z-80 code, the Z-80 assembly language modules must be linked at a specific location, i.e. 9100H. This requires that special instructions be applied when executing the linking function.

In addition, the FORTH/Z-80 interface program, BADJR, must be listed as the first program to be linked. Therefore, the command used to link the Z-80 code correctly is:

```
    LINK BADJR[9100],ATRB,BLCK,BOOL,CONV,IMED,IONS,MATH,
    MIOS,RADX,RELN,STOR
```

This will produce a symbol file and an execution file BADJR.SYM and BADJR.COM, respectively.

4. LOADING A NEW FORJR SYSTEM

The FORJR system is comprised of two distinct programs, FORTH and Z-80 code. Both programs must be in memory simultaneously in order to create the new FORJR system. DDT is used to load both programs.

To begin with, a basic FORTH system is loaded via DDT. The current FORTH system used is called HAZEL.COM, a FORTH version for the HAZELTINE 1500 CRT. The command to load HAZEL.COM is:

```
                    DDT HAZEL.COM
```

DDT will load the FORTH code into low memory begining at address 100H.

After the FORTH code is loaded, the Z-80 assembly languge module, BADJR.COM, that has been linked as above must be loaded at address 9100H. The DDT commands I (for INPUT) and R (for READ) are used. When DDT loads programs, the loader offsets the load address by 100H. Therefore you must specify a load address that is 100H LESS than the actual address desired. Therefore, the commands for inputting and reading BADJR.COM code are:

```
IBADJR.COM
R9000
```

This will load the Z-80 code begining at 9100H.

After both FORTH and Z-80 programs have been loaded, invoke the FORTH system via:

```
G100
```

5. TESTING THE MODIFIED FORJR SYSTEM

When the FORTH system comes up after G100, none of the FORJR commands exist in the FORTH dictionary. Therefore, you must change to the BADJR user screen file via the USING command:

```
USING BADJSCR
```

The FORJR dictionary entries can be loaded begining with screen number nine via:

```
9 LOAD
```

When all the BADJR screens have been loaded, testing of the modified system can begin. If testing is successful, a new FORJR.COM file can be created with all the desired features of the new FORJR system in the protected dictionary space.

6. BUILDING A NEW FORJR.COM FILE

The whole FORJR system is closely tied to addresses which implies that the FORTH dictionary used must be a specific size. The dictionary size of the basic FORTH system loaded as above for testing must be expanded to accomodate the necessary addressing capabilities. In the file FORTH.SCR, screen # 119 contains the necessary commands to expand the dictionary size. Change to the FORTH.SCR file via:

```
USING FORTH.SCR
```

Load screen # 119 via:

```
119 LOAD
```

This will automatically execute the commands to expand the dictionary. The program will ask two questions:

(1) Size of FORTH area (KBYTES):

To which your response MUST be:

```
36
```

(2) Enter # of screens to buffer:

To which your response MUST be:

```
4
```

The program will then expand the dictionary size to 11977 bytes, and also execute a COLD which deletes all but the system dictionary entries. You must reload the FORJR screen contents. Switch back to the FORJR screen file via:

```
USING BADJSCR
```

Then reexecute 9 LOAD. After the load is complete, you have to create a new .COM file. Screen # 3 in the BADJSCR file is used for this purpose. Execute this via:

```
3 LOAD
```

The system will exit from FORTH back to CP/M and tell you to enter SAVE 94 filename.COM. However, in order to establish the correct file size, you MUST enter:

```
SAVE 128 filename.COM
```

This will create a temporary file with 256 records that will be used to create a final updated version of the FORJR.COM system.

The new FORJR.COM file is comprised of the temporary file created above combined with a "filler" file, BOTOM.COM that is 32 records long, and also the BADJR.COM file which is 125 records long. All three files are copied into a single file via the CP/M Peripheral Interchange Program, PIP. The actual PIP command is:

```
PIP FORJR.COM=filename.COM,BOTOM.COM,BADJR.COM
```

After the copy is complete, you may begin using FORJR as indicated in paragraph 1 of this guide.

APPENDIX B

FORTH SCREEN CONTENTS

This appendix contains the FORTH screens used in FORJR. Screens 9 through 26 contain the instructions. Examples of FORJR programs are contained in screens 27 through 30.

```
   Screen # 12
( LOGICAL OPERATORS & CHARACTERISTIC FUNCTIONS )
: BAND 7 ZBADJR RSTINH ;          ( A1 A2 A3 --> BOOL AND IN A3 )
: BOR  8 ZBADJR RSTINH ;          ( A1 A2 A3 --> BOOL OR IN A3 )
: BXOR 9 ZBADJR RSTINH ;          ( A1 A2 A3 --> BOOL XOR IN A3 )
: BNOT 10 ZBADJR RSTINH ;         ( A1 A2 --> BOOL NOT IN A2 )

( CHARACTERISTIC FUNCTIONS )
: ATOM? 11 ZBADJR RSTINH ;        ( A1 A2 --> BOOL ANS IN A2 )
: NIL?  12 ZBADJR RSTINH ;        ( A1 A2 --> BOOL ANS IN A2 )
: SYMBOL? 13 ZBADJR RSTINH ;      ( A1 A2 --> BOOL ANS IN A2 )
: NUMBER? 14 ZBADJR RSTINH ;      ( A1 A2 --> BOOL ANS IN A2
: BOOLEAN? 15 ZBADJR RSTINH ;     ( A1 A2 --> BOOL ANS IN A2 )
: EMPTY? 16 ZBADJR RSTINH ;       ( A1 A2 --> BOOL ANS IN A2 )
: SEQUENCE? 17 ZBADJR RSTINH ;    ( A1 A2 --> BOOL ANS IN A2 )

-->

   Screen # 13
( IMMEDIATE NUMBER & SYMBOL GENERATOR )
HEX FORTH DEFINITIONS
100 ALLOT          ( allocate the string stack )
HERE CONSTANT $0 ( fixed base of $STK )
$0 VARIABLE $P    ( $P returns address of var with $STK ptr )
: $DROP $P @ DUP @ + 2+ $P ! ;     ( drop top string )
: $@ DUP >R $P @ SWAP - SWAP OVER R CMOVE 2 - R> OVER ! $P ! ;
: $. $P @ DUP 2+ SWAP @ ; ( --> STRINGADDR N )
: NUM $. 15 ZBADJR $DROP ; ( CREATES AN IMMEDIATE NUMBER )
: SYM $. 16 ZBADJR $DROP ; ( CREATES AN IMMEDIATE SYMBOL )


-->



   Screen # 14
( IMMEDIATE NUMBER & SYMBOL GENERATOR, continued )
( PUTS AN IMMEDIATE SYMBOL INTO CURRENT FRAME )
: (")  R DUP 2+ SWAP @ ( moves in-line string to $STK )
  DUP 2+ R> + >R $@ ;
: "      ( if compiling emplace an in-line string to be )
         ( moved to string stack at execution time, else )
         ( put enclosed string on string stack. )
  22 STATE @
  IF COMPILE (") 0 C, WORD HERE C@ -1 ALLOT DUP , ALLOT
  ELSE 0 C, WORD HERE C@ -1 ALLOT HERE !
         HERE DUP 2+ SWAP @ $@
         ENDIF
  STATE @
  IF COMPILE SYM
  ELSE SYM ENDIF ; IMMEDIATE
  -->
OK
```

```
   Screen # 15
 0 ( IMMEDIATE NUMBER & SYMBOL GENERATOR, continued )
 1 ( PUTS AN IMMEDIATE NUMBER ATTRIBUTE INTO FRAME )
 2 : #  ( if compiling embrace an in-line string to be )
 3        ( moved to string stack at execution time, else )
 4        ( put enclosed string on string stack. )
 5   20 STATE @
 6   IF COMPILE (") 0 C, WORD HERE C@ -1 ALLOT DUP , ALLOT
 7   ELSE 0 C, WORD HERE C@ -1 ALLOT HERE !
 8        HERE DUP 2+ SWAP 3 $@
 9        ENDIF
10   STATE @
11   IF COMPILE NUM
12   ELSE NUM ENDIF ; IMMEDIATE
13 DECIMAL
14 -->
15

   Screen # 16
 0 ( ATTR STACK & SEQ MANIPULATION ROUTINES )
 1 : A1 1 STRINH ; : A2 2 STRINH ; : A3 3 STRINH ; : A4 4 STRINH ;
 2 : A5 5 STRINH ; : A6 6 STRINH ; : A7 7 STRINH ; : A8 8 STRINH ;
 3 : A9 9 STRINH ; : A10 10 STRINH ; : A11 11 STRINH ;
 4
 5 ( SEQUENCE MANIPULATION ROUTINES )
 6 : SL 23 ZBADJR ; ( I A1 --> NEWATTR ) ( IMMEDIATE SELECT
 7                   ( A1 MUST BE TOP ATTRIBUTE IN FRAME )
 8 : << SETBAS ;     ( BEGIN SEQUENCE CONSTRUCTION )
 9 : >> 25 ZBADJR ; ( END   SEQUENCE CONSTRUCTION )
10 : MERGE   SETBAS ; ( MERGES SEQ X...) INTO SEQ Z )
11 : CLSMER  29 ZBADJR ; ( ENDS MERGE OPERATION )
12 -->
13
14
15

   Screen # 17
 0 ( CONVERSION & PRIMITIVE ROUTINES )
 1 : ID       30 ZBADJR RSTINH ;   ( A1 A2 --> ) ( A2 = A1 )
 2 : SYMSEQ   33 ZBADJR RSTINH ;   ( A1 A2 --> ) ( A2 IS SEQUENCE )
 3 : SEQSYM   34 ZBADJR RSTINH ;   ( A1 A2 --> ) ( A2 IS SYMBOL )
 4 : SEQNUM   35 ZBADJR RSTINH ;   ( A1 A2 --> ) ( A2 IS NUMBER )
 5 : NUMSYM   36 ZBADJR RSTINH ;   ( A1 A2 --> ) ( A2 IS SYMBOL )
 6
 7 ( PRIMITIVE ROUTINES )
 8 ( A1 MUST be sequence. In DL & DR, A2 can be any object )
 9 ( In SEL & SER, A2 MUST be a numeric attribute )
10 : RV       37 ZBADJR RSTINH ;   ( A1 A2 --> A2 IS REV OF A1 )
11 : DL       38 ZBADJR RSTINH ;   ( A1 A2 A3 --> A2 IS DL OVER A1 )
12 : DR       39 ZBADJR RSTINH ;   ( A1 A2 A3 --> A2 IS DR OVER A1 )
13 : SEL      40 ZBADJR RSTINH ;   ( A1 A2 A3 --> A3 IS ELT OF A1 )
14 : SER      41 ZBADJR RSTINH ;   ( A1 A2 A3 --> A3 IS ELT OF A1 )
15 -->
OK
```

```
   Screen # 18
( RELATIONAL OPERATORS )
: EQ? 43 ZBADJR RSTINH ; ( A1 A2 A3 --> BOOL ANS IN A3 )
: NE? 44 ZBADJR RSTINH ; ( A1 A2 A3 --> BOOL ANS IN A3 )
: LT? 45 ZBADJR RSTINH ; ( A1 A2 A3 --> BOOL ANS IN A3 )
: LE? 46 ZBADJR RSTINH ; ( A1 A2 A3 --> BOOL ANS IN A3 )
: GT? 47 ZBADJR RSTINH ; ( A1 A2 A3 --> BOOL ANS IN A3 )
: GE? 48 ZBADJR RSTINH ; ( A1 A2 A3 --> BOOL ANS IN A3 )
-->

   Screen # 19
( ARIMTHMETIC PRIMITIVES )
( In AD, SB, ML, DV, and MD: A1 & A2 are numeric, A3 is synth )
( In INT, AB, NG: A1 is numeric, A2 is synthesized )
: AD 49 ZBADJR RSTINH ;  ( A1 A2 A3 --> A3 = A1 + A2 )
: SB 50 ZBADJR RSTINH ;  ( A1 A2 A3 --> A3 = A1 - A2 )
: ML 51 ZBADJR RSTINH ;  ( A1 A2 A3 --> A3 = A1 * A2 )
: DV DEFLOC ( A2 # 0 A4 ) EQ?
     ( A4 ) QUES
     IF CR ." ZERO DIVIDE PROHIBITED "
     ELSE 52 ZBADJR ENDIF RSTINH ;
: INT 55 ZBADJR RSTINH ; ( A1 A2 --> A2 = INTEGER OF A1 )
: MD DEFLOC DEFLOC DEFLOC DEFLOC ( A1 A2 A3 --  A3 = A1 MOD A2 )
     ( A2 A4 ) INT ( A1 A4 A5 ) DV
     ( A5 A6 ) INT ( A4 A6 A7 ) ML ( A1 A7 A3 ) SB RSTINH ;
: AB 53 ZBADJR RSTINH ;  ( A1 A2    --> A2 = ABS[A1] )
: NG 54 ZBADJR RSTINH ;  ( A1 A2    --> A2 = - A1 )     -->

   Screen # 20
( KEYBOARD INPUT ROUTINE )
HEX
: $INPUT
    PAD DUP
      BEGIN KEY DUP 08 =
      IF >R 2DUP = R> SWAP
        IF DROP 0       ( if 1st char, ignore )
        ELSE DROP 08 EMIT BL EMIT 08 EMIT 1- 0 ENDIF
      ELSE DUP 0D =
        IF DROP BL EMIT 1
        ELSE DUP EMIT OVER C! 1+ 0 ENDIF
      ENDIF
      UNTIL
    OVER - $@ ;
DECIMAL
-->
OK
```

```
Screen # 21
( I/O PRIMITIVE I/O )


: PRNUM 57 ZBADJR RSTINH ;
: PRSYM 59 ZBADJR RSTINH ;
: PRBUL 61 ZBADJR RSTINH ;
: RDSYM CR ." INPUT SYMBOL: "  $INPUT SYM ;


 ( MISCELLANEOUS INSTRUCTIONS )
: POPINH 70 ZBADJR ;
: COLECT 71 ZBADJR ;
: LENGTH 72 ZBADJR ;
: RSTBAS 73 ZBADJR ;


```

```
  Screen # 22
( MEMORY DUMP ROUTINES & INITSTORE )
0    VARIABLE COUNTER
: DUMP CR HEX DUP U. 0 COUNTER ! 80 0 DO
     COUNTER @ 15 > IF 0 COUNTER ! DUP CR U. ENDIF
  1 COUNTER +! DUP DUP C@ SWAP 1+ C@ 1 .R 1 .R SPACE 2 + LOOP
  DROP DECIMAL ;
: DUMPINH CR CR ." DUMP OF FRAME STACKS " CR INHSTK DUMP ;
: DUMPNOD CR CR ." DUMP OF NODES " CR NODES DUMP ;
: DUMPSTR CR CR ." DUMP OF STRINGSPACE " CR
     STRSPACE HEX DUP CR U. 0 COUNTER ! 100 0 DO
     COUNTER @ 7 > IF 0 COUNTER ! DUP CR U. ENDIF
     1 COUNTER +! DUP C@ 3 .R 1+ DUP C@ 3 .R 1-
  LOOP DROP DECIMAL CR ;

: INITSTORE 69 ZEADUR EOL ;  ( INITIALIZE STORAGE AREAS )
-->

  Screen # 23
( AUTO FRAME SETUP & RECURSIVE SETUP ROUTINES )
( AUTO FRAME SETUP ROUTINES
0 VARIABLE ATTCOUNT  ( COUNTER VARIABLE FOR ATTRIBUTES )
0 VARIABLE LOCCOUNT  ( COUNTER VARIABLE FOR LOCAL ATTRIBUTES )
: << 0 LOCCOUNT ! 0 ATTCOUNT ! ; IMMEDIATE ( SETS COUNTS TO 0 )
: >> ;   ( DUMMY WORD - FOR READIBILITY ONLY )
: ^^ <BUILDS LOCCOUNT DUP @ 1+ SWAP !  ( ^^ xxx = SYNTH ATTR )
        ATTCOUNT DUP @ 1+ DUP , SWAP ! DOES> @ STKINH ;
: ^       ( ^ yyy = INHERITED ATTR )
  <BUILDS ATTCOUNT DUP @ 1+ DUP , SWAP ! DOES> @ STKINH ;
: DEFLOCS 0 DO DEFLOC LOOP ; ( SETS UP n SYNTHESIZED ATTRS )
: LOC ' LIT CFA , LOCCOUNT @ , ' DEFLOCS CFA , ; IMMEDIATE
: ; ; ( "SEPERATES INHERITED ATTRIBUTES FROM SYNTHESIZED" )
( FORTH RECURSIVE SETUP ROUTINES )
: DEFINE 2+ LATEST PFA SWAP ! ; ( RECURSIVE SETUP )
: BADJP <BUILDS 0 , DOES> @ EXWORD ! ; ( RECURSIVE SETUP ) -->
OK
```

```
Screen # 24
( SEQUENCE INFORMATION ROUTINES )
: OBAS PTR 12 + @ ;          ( -- ADDR OF BASE OF PREVIOUS FRAME )
: BAS PTR 14 + @ ;           ( -- ADDR OF BASE OF FRAME )
: TOP PTR 16 + @ ;           ( -- ADDR OF TOP  OF FRAME
: (** BAS OBAS 2+ - 2 / STRINH ;   ( STACK TOP ATT FROM PREV FRM )
: TYP 2+ C@ ;                ( STRING ADDR -- STRING TYPE
: NXT 3 + @ ;     ( STRING ADDR -- OFFSET TO NEXT STR )

: LEN NXT 5 - ;              ( STRING ADDR -- STRING LENGTH )
: SQLN DUP                   ( STRING ADDR -- SEQUENCE LENGTH )
     TYP 224 = IF LEN 2
                 ELSE CR ." STRING NOT SEQUENCE " DROP ENDIF ;

: FINDTYP 1 - 4 * NODES + @ DUP  ( ATTR IDX -- ATTR TYPE )
     0= IF ELSE TYP 1 ENDIF ;
: FINDIDX 1 - 2 * BAS 2 + + @ ;  ( FRM ATTR# -- ATTR IDX ) -->

Screen # 25
( STRINGSPACE INFO & FRAME DUMP ROUTINES )
: STRADDR ( A1 --> ADDRESS OF STRING ON TOP. RESETS FRAME )
           BAS 2+ @ 1 - 4 * NODES + @ RETINH ;

: SEQADD BAS 2+ @ 1 - 4 * NODES + @ ; ( --> ADDR OF TOP SEQ )

: SP ( N1 --  ELT FROM SEQ SELECTED FROM RIGHT OF SEQ )
  SEQADD  SQLN SWAP - 1+ SL ; ( SEQUENCE MANIPULATION ROUTINE )

: RDNUM ( CREATES IMMEDIATE NUMERIC ATTRIBUTE FROM KEYBOARD )
     CR ." INPUT NUMBER > " $INPUT ** ( NUM # ! >** ) ML ;

: SEQLEN STRADDR SQLN ; ( A1 --> LENGTH OF SEQUENCE )


-->

Screen # 26
( FRAME PRINTOUT ROUTINE )
HEX 0 VARIABLE ATT#
: ATTSTK ( ATT# @ STKINH ) ;
: PRINTIT
    DUP C1 = IF ." NEG NUMBER " ATTSTK PRNUM ENDIF
    DUP C2 = IF ." POS NUMBER " ATTSTK PRNUM ENDIF
    DUP D0 = IF ." SYMBOL " CR  ATTSTK PRSYM CR ENDIF
        E0 = IF ." SEQUENCE " CR ATTSTK SEQLEN
                ." LENGTH = " . CR ENDIF ;
: FRAME ( --> PRINTS CURRENT FRAME ADDR & ATTRIBUTES )
     HEX TOP BAS DUP CR CR ." FRAME ADDR: " U. CR CR 2+ 2DUP =
    IF CR ." FRAME EMPTY " DROP DROP ELSE - 2 / 1+
    1 DO I DUP ATT# ! FINDIDX FINDTYP
         DECIMAL ." ATTR # " I . ." : " HEX
         0= IF ." NOT DEFINED " CR
         ELSE PRINTIT ENDIF LOOP ENDIF CR DECIMAL ; DECIMAL
OK
```

THE FOLLOWING SCREENS ARE SAMPLE PROGRAMS FOR FORJR

FACTOR IS A RECURSIVE FACTORIAL PROGRAM
TO US FACTOR, LOAD SCREEN 27 FROM BADJSCR.SCR VIA:

```
        27 LOAD
```

THEN TYPE:

```
        FACTOR
```

```
  Screen # 27
( RECURSIVE FACTORIAL PROGRAM )
(( ^ X ^ Y ^ Z ^^ A ^^ B ^^ C ))
BADJR FACT
: LINE1 [ ' FACT DEFINE ]  ( REDEFINES LINE1 AS FACT )
     LOC   ( DEFINES LOCAL ATTRIBUTES )
     ( Y # 1 : A ) LE? ( A ) QUES
     IF ( # 1 X : Z ) ML
        ELSE
            ( Y # 1 : B ) SB
            ( X   Y : C ) ML
            ( C   B : Z ) FACT
        ENDIF
      EOL ;
: FACTOR CR ." MAXIMUM INPUT = 80 " CR
  INITSTORE ( DEFLOC ) ( # 1 RDNUM : X ) FACT CR
  ." ANSWER = " ( X ) PRNUM EOL ;
OK
```

```
CALC IS AN EXAMPLE OF A HAND CALCULATOR

TO USE CALC, LOAD SCREEN 28 FROM BADGER.BIR VIA:

          28 LOAD

THEN TYPE:

                         CALC

CALC OPERATES UNTIL AN ILLEGAL SYMBOL IS ENTERED
```

```
   Screen # 28
 ( ATTRIBUTE NAMING FOR HAND CALCULATOR )
 {{ ^ A ^ B ^ C ^ D ^^ E }}  ( DEFINE VARIABLE NAMES
 : POP&DEF POPINH LOC ;
 : STK5 LOC { A B ! E } ;
 : PR5&SLIDE CR ." ANSWER = "
      { E ! PRNUM { E } SLIDE 0 ;
 -->
```

```
   Screen # 29
 ( HAND CALCULATOR, continued )
 : CALC CR ." CALCULATOR ON " INITSTORE { # 0 }
     CR ." VALID SYMBOLS: + - * / C "
     BEGIN  { A FDNUM FDS.M LOC } ( VARIABLES A -> D )
      { C " +" ! D } EQ?
      { D } QUES IF STK5 AD PR5&SLIDE            ( ADD )
        ELSE POP&DEF { C " -" ! D } EQ?
        { D } QUES IF STK5 SB PR5&SLIDE         ( SUBTRACT )
          ELSE POP&DEF { C " *" ! D } EQ?
          { D } QUES IF STK5 ML PR5&SLIDE       ( MULTIPLY )
            ELSE POP&DEF { C " /" ! D } EQ?
            { D } QUES IF STK5 DV PR5&SLIDE    ( DIVIDE )
              ELSE POP&DEF { C " C" ! D } EQ?
              { D } QUES IF FSTINH { # 0 } 0   ( CLEAR )
              ELSE CR ." CALCULATOR OFF " 1
          ENDIF ENDIF ENDIF ENDIF ENDIF UNTIL ;
OK
```

```
SOS IS A RUNNING SUM OF SQUARES PGM

TO USE SOS, LOAD SCREEN 30 FROM BADJSCR.SCR VIA:

          30 LOAD

THEN TYPE:

          SOS


SOS RUNS UNTIL THE NUMBER ZERO IS ENTERED
```

```
    Screen # 30
  0 ( INTERACTIVE SUM OF SQUARES PROGRAM )
  1 << ^ V ^ W ^^ X ^^ Y ^^ Z >> ( SET UP ATTRIBUTE NAMES )
  2                               ( X , Y , Z  ARE LOCAL ATTR. )
  3 : SOS
  4   CR ." ENTER DESIRED NUMBER. PROGRAM ENDS WHEN ZERO ENTERED "
  5   CR INITSTORE < # 0 >
  6      BEGIN RDNUM LOC ( DEFINE 3 LOCAL VARIABLES )
  7         < W W ! X > ML
  8         CR ." INPUT NUMBER SQUARED = " CR < X > PRNUM
  9         < V X ! Y > AD
 10         CR ." RUNNING TOTAL = " CR < Y > PRNUM
 11         < W # 0 ! Z > EQ?
 12         < Z > QUES
 13         IF 1 ( QUIT )
 14         ELSE < Y > SLIDE 0 ( DO NOT QUIT ) ENDIF
 15      UNTIL ;
OK
```

APPENDIX C

Z-80 SOURCE LISTINGS

This appendix contains the source listings of the Z-80 modules used in FORJR. Table C.1 provides a short description of the responsibilities of each module.

TABLE C.1

Z-80 SOURCE CODE FILES AND MAJOR SUBROUTINES

BADJR: FORTH/Z80 Interface Program and Jump Table

ATRB: Attribute Frame Management

SETINH	STKINH	RSTINH	DEFLOC
QUES	PSHINH	POPINH	SETBAS
RSTBAS			

BLCK: Storage Initialization, Storage Areas

INITSTOR	SAVREG	RSTREG

BOOL: Logical Operations, Predicates

BAND	BOR	BXOR	BNOT
ATOM?	NIL?	SYMBOL?	NUMBER?
BOOLEAN?	EMPTY?	SEQUENCE?	

CONV: Conversions, Sequence Manipulations

ID	SYMSEQ	SEQNUM	NUMSYM
RV	TR	DL	DR
SEL	SER		

IMED: Immediate Instructions

NUM	SYM	SL	LN
CONS	MERGE		

IONS: Input/Output Instructions

PRNUM	PRSYM	PRBUL

MATH: Arithmetic Instructions

AD	SB	ML	DV
NG	ABS		

RADX: Radix Conversion

BCDASC	ASCBCD	HEXASC	ASCHEX
HEXBCD	BCDHEX		

TABLE C1 (CONTINUED)

RELN: Relational Instructions

EQ?	NE?	LT?	LE?
GT?	GE?		

STOR: Storage Management

SLIDE	COLECT	GC	ALLOC
FETCH	GETNOD		

MACROS: Macro File Used In Original Zbadjr

EQATMO: Equate File Defining Constants

```
;
; 31 OCT 83 - ORIGINAL
; 22 NOV 83 - ADDED LENIMM
; 28 NOV 83 - INSTALLED REFERENCES TO STORAGE AREAS
; 13 DEC 83 - ADDED STORAGE AREAS FOR FORTH I/O
; 14 DEC 83 - BEGAN REMOVING DEAD WOOD STORAGE
; 06 JAN 84 - INSTALLED RSTBAS IN JUMPTABLE
; 13 JAN 84 - REMOVED TR
;
        TITLE BADJR A/O 13 JAN 84
;
EXTERNAL AB,AD,ATOM?
EXTERNAL BAND,BNOT,BOOLEAN?,BOR,BXOR
EXTERNAL COLECT,CONIMM
EXTERNAL DEFLOC,DL,DR,DV
EXTERNAL EMPTY?,EQ
EXTERNAL FORRST,FORSAV
EXTERNAL GE,GT
EXTERNAL ID,INITST,INPBUF
EXTERNAL INT
EXTERNAL LE,LT
EXTERNAL MERIMM,ML
EXTERNAL NE,NG,NIL?,NUMBER?,NUMIMM,NUMSYM
EXTERNAL POPINH,PRBUL,PRNUM,PRSYM,PSHINH
EXTERNAL QUES
EXTERNAL RSTBAS,RSTINH,RSTREG,RV
EXTERNAL SAVREG,SB,SEL,SELIMM,SEQNUM,SEQSYM,SEQUENCE?,SER
EXTERNAL SETBAS,SETINH,SLIDE,STKINH,LENIMM
EXTERNAL SYMBOL?,SYMIMM,SYMSEQ
;
EXTERNAL PTR,HDR,INHSTK,NODLST,NODES
GLOBAL BTTABLE,FORRTN,PRFLAG,PRADDR,PRNUMB,BDOSFLG
GLOBAL  BADENTRY
;
        MACLIB EQATMO
        EQUATES
;
;***********************************
; THIS IS THE ROUTINE TO INTERFACE
; FORTH WITH THE Z-80 ZBADJR ROUTINES
;***********************************
;
        .SALL
;
BADENTRY:
       JP       STRT            ; JUMP AROUND STORAGE AREAS
;
;*******************************************
; REFERENCES TO MEMORY STORAGE AREAS
;*******************************************
;
        DW      PTR             ; ADDRESS OF POINTER
        DW      HDR             ; ADDRESS OF HEADER
        DW      INHSTK          ; ADDRESS OF INHERITANCE ST
        DW      NODLST          ; ADDRESS OF NODELIST
        DW      NODES           ; ADDRESS OF STRINGSPACE
        DW      PRFLAG          ; ADDRESS OF PRINT REQUEST
        DW      PRADDR          ; ADDRESS OF BEGINNING OF
        DW      PRNUMB          ; ADDR OF # OF BYTES TO PR
        DW      BDOSFLG         ; ADDR OF SYSTEM PRINTOUT
;
; SAVE FORTH ENVIRONMENT
;
STRT:   CALL    FORSAVE         ; SAVE FORTH REGISTERS
        LD      HL,0            ;
        LD      (PRFLAG),HL     ; ZERO OUT PRINT FLAG
        POP     DE              ; SAVE RETURN ADDRESS TO F
        LD      (FORRTN),DE     ; SAVE FORTH RETURN ADDRES
        POP     HL              ; GET INDEX INTO JUMP TABL
;
; SET INDEX INTO JUMPTABLE
;
        LD      DE,BTTABLE
        ADD     HL,DE
        LD      DE,RETADD
        PUSH    DE
        JP      (HL)
;
RETADD:
        LD      DE,(FORRTN)     ; RESTORE FORTH RETURN ADD
        PUSH    DE
        CALL    FORRST          ; RESTORE FORTH REGISTERS
        RET
;
; JUMP TABLE FOR ZBADJR ROUTINES
;
BTTABLE:
;
; FRAME MANIPULATION ROUTINES
;
        JP      SETINH          ; 0. SETS A NEW BAS, OBAS
        JP      STKINH          ; 1. STACKS ATTRIBUTES ONT
        JP      RSTINH          ; 2. RESETS BAS, OBAS TO P
        JP      SETBAS          ; 3. SETS NEW BAS
        JP      DEFLOC          ; 4.  ( # --> ) DEFINES LO
        JP      QUES            ; 5. DETERMINES STATUS OF
        JP      SLIDE           ; 6. SLIDES CURRENT FRAME
;
; LOGICAL OPERATORS
;
        JP      BAND            ; 7. BOOLEAN "AND"
        JP      BOR             ; 8. BOOLEAN "OR"
        JP      BXOR            ; 9. BOOLEAN "XOR"
        JP      BNOT            ; 10. BOOLEAN "NOT"
;
; CHARACTERISTIC FUNCTIONS
```

```
;
        JP      ATOM?           ; 11. IS OBJECT AN ATOM?
        JP      NIL?            ; 12. CHECKS FOR NIL SEQUEN
        JP      SYMBOL?         ; 13. IS ATTRIBUTE A SYMBOL
        JP      NUMBER?         ; 14. IS ATTRIBUTE A NUMBE
        JP      BOOLEAN?        ; 15. IS ATTRIBUTE BOOLEAN
        JP      EMPTY?          ; 16. CHECKS FOR EMPTY SYM
        JP      SEQUENCE?       ; 17. CHECKS FOR SEQUENCE
FINITE?: JP     EXIT            ; 18. RESERVED FOR FINITE?
STREAM?: JP     EXIT            ; 19. RESERVED FOR STREAM?
DRY?:   JP      EXIT            ; 20. RESERVED FOR DRY?
;
; ATTRIBUTE BUILDING ROUTINES
;
        JP      NUMIMM          ; 21. MAKES A NUMERIC ATTR
        JP      SYMIMM          ; 22. MAKES A SYMBOLIC ATT
        JP      SELIMM          ; 23. IMMED. SEL FROM A SE
        JP      SELIMM          ; 24. IMMED. SER FROM A SE
        JP      CONIMM          ; 25. ENDS SEQ CONSTRUCTOR
CATIMM: JP      EXIT            ; 26. RESERVED FOR CATIMM
HEAD:   JP      EXIT            ; 27. RESERVED FOR HEAD
TAIL:   JP      EXIT            ; 28. RESERVED FOR HEAD
        JP      MERIMM          ; 29. ENDS SEQ MERGE FUNCT
;
; CONVERSION ROUTINES
;
        JP      ID              ; 30. MAKES AND IDENTICAL
SEQSTR: JP      EXIT            ; 31. RESERVED FOR SEQSTR
STRSEQ: JP      EXIT            ; 32. RESERVED FOR STRSEQ
        JP      SYMSEQ          ; 33. MAKES A SEQ FROM A
        JP      SEQSYM          ; 34. MAKES A SYMBOL FROM
        JP      SEQNUM          ; 35. MAKES A NUMBER FROM
        JP      NUMSYM          ; 36. MAKES A SYMBOL FROM
;
; SEQUENCE MANIPULATION ROUTINES
;
        JP      RV              ; 37. MAKES A REVERSE SEQU
        JP      DL              ; 38. DIST. LEFT OVER A SE
        JP      DR              ; 39. DIST. RIGHT OVER A S
        JP      SEL             ; 40. DOES PRIM SELECT FRO
        JP      SER             ; 41. DOES PRIM SELECT FROM
TR:     JP      EXIT            ; 42. RESERVED FOR TR
;
; IN THE FOLLOWING FUNCTIONS, A3 IS RETURNED AS A BOOLEAN
; DEPENDING UPON THE RESULT OF THE COMPARISON OF A1 AND A2
;
        JP      EQ              ; 43. CHECKS IF A1 = A2
        JP      NE              ; 44. CHECKS IF A1 /= A2
        JP      LT              ; 45. CHECKS IF A1 < A2
        JP      LE              ; 46. CHECKS IF A1 <= A2
        JP      GT              ; 47. CHECKS IF A1 > A2
        JP      GE              ; 48. CHECKS IF A1 >= A2
;
```

```
; ARITHMETIC FUNCTIONS
; IN THE FOLLOWING ZBADJR FUNCTIONS, A3 IS RETURNED AS A N
; ATTRIBUTE WITH THE RESULT OF THE ARITHMETIC OPERATION
;
          JP      AD              ; 49. A3 = A1 + A2
          JP      SB              ; 50. A3 = A1 - A2
          JP      ML              ; 51. A3 = A1 * A2
          JP      DV              ; 52. A3 = A1 / A2
          JP      AB              ; 53. A2 = ! A1 ! ( ABSOLU
          JP      NG              ; 54. A2 = - A1 ( NEGATION
          JP      INT             ; 55. A2 = INTEGER VALUE O
;
; I/O FUNCTIONS
;
RDNUM:    JP      EXIT            ; 56. RESERVED FOR RDNUM
          JP      PRNUM           ; 57. PRINTS INTEGER VALUE
RDSYM:    JP      EXIT            ; 58. RESERVED FOR RDSYM
          JP      PRSYM           ; 59. PRINTS SYMBOL FROM A
RDBUL:    JP      EXIT            ; 60. RESERVED FOR RDBUL
          JP      PRBUL           ; 61. PRINTS TRUE/FALSE OF
;
; HIGHER LEVEL FUNCTIONS
;
WHILE1:   JP      EXIT            ; 62. RESERVED FOR WHILE1
WHILE2:   JP      EXIT            ; 63. RESERVED FOR WHILE2
APPLY1:   JP      EXIT            ; 64. RESERVED FOR APPLY1
APPLY2:   JP      EXIT            ; 65. RESERVED FOR APPLY2
STKWLD:   JP      EXIT            ; 66. RESERVED FOR SKTWLD
INSERT:   JP      EXIT            ; 67. RESERVED FOR INSERT
IOSEL:    JP      EXIT            ; 68. RESERVED FOR INSERT
;
; MISCELLANEOUS COMMANDS
;
          JP      INITSTORE       ; 69. INITIALIZE STORAGE A
          JP      POPINH          ; 70. REMOVES TOP ATTR FRO
          JP      COLECT          ; 71. COMPACTS STRING AND
          JP      LENIMM          ; 72. RETURNS LENGTH OF SE
          JP      RSTBAS          ; 73. RESETS BAS
EXIT:     RET                     ; USED FOR RESERVED ROUTIN
FORRTN:   DW      1
PRFLAG:   DW      1               ; 12/13 PRINT FLAG
PRADDR:   DW      1               ; 12/13 START OF PRINT ADD
PRNUMB:   DW      1               ; 12/13 # OF BYTES TO PRIN
BDOSFLG:  DW      0               ; 12/15 FLAG FOR BDOS CALL
END
```

```
;
;  6 OCT 83
; 28 OCT 83 - CHANGED STKINH (POP BC)
; 23 NOV 83 - CHANGED QUES TO SUPPORT FORTH
; 14 DEC 83 - REMOVED ALL EXTRANEOUS INSTRUCTIONS/STORAGE
;
        TITLE ATRB A/O 14 DEC 83
;
;ROUTINES TO HANDLE INHERITED ATTRIBUTE STACK
;AND TO PASS ATTRIBUTES TO BADJR FUNCTIONS
;ALSO INCLUDES "QUES" THE CONDITIONAL LINE
;ROUTINE
        .Z80
        .SALL
;       GLOBAL TATRB
GLOBAL SETINH,STKINH,RSTINH,DEFLOC
GLOBAL SETBAS,RSTBAS
GLOBAL INH.1,INH.2,INH.3
GLOBAL INH.4
GLOBAL PSHINH,POPINH
GLOBAL GETATR,ALOSYN,ATR
GLOBAL QUES
;
;
EXTERNAL HDR,PTR,FETCH,ALLOC,GETNOD
EXTERNAL SAVREG,RSTREG,STLUP1,LUP1
EXTERNAL PRLINE
;
                        .XLIST
        MACLIB EQATMO
        EQUATES
                        .LIST
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SETINH:
INH.1:
;SETS A NEW BOTTOM FOR INHSTK WHEN A LINE IS DEFINED
        CALL SAVREG
        LD HL,(BAS+PTR) ;SAVE OLD (BAS+PTR)
        LD (OBAS+PTR),HL
        LD DE,(PTR+TOP) ;NEW BAS=OLD TOP
        LD (PTR+BAS),DE
        LD HL,PTR+OBAS  ;PUSH OLD BAS
        LDI
        LDI
        LD (TOP+PTR),DE
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
STKINH:
```

```
INH.2:
;STACKS UP ONE MORE INDEX FROM CALLER'S LIST
        CALL SAVREG
        POP     HL              ; SAVE RETURN ADDRESS
        POP     BC
        PUSH    HL              ; RESTORE RETURN ADDRESS
        LD IX,(OBAS+PTR)
        LD IY,(TOP+PTR)
        ADD IX,BC       ;GET IDX FROM OLD STK
        ADD IX,BC
        LD C,(IX)       ;SAVE THE IDX
        LD B,(IX+1)
        LD (IY),C
        LD (IY+1),B
        INC IY
        INC IY
        LD (TOP+PTR),IY ;RESET TOP
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
RSTINH:
INH.3:
;SETS BOTTOM BACK TO CALLER'S BOTTOM
        CALL SAVREG
        LD HL,(BAS+PTR)
        LD (TOP+PTR),HL
        LD DE,BAS+PTR
        LDI
        LDI
        LD HL,(BAS+PTR)
        LD DE,OBAS+PTR
        LDI
        LDI
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SETBAS:
;SETS NEW BAS WITHOUT SETTING NEW OBAS
        CALL SAVREG
        LD DE,(TOP+PTR)
        LD HL,BAS+PTR
        LDI
        LDI
        LD (TOP+PTR),DE
        DEC DE
        DEC DE
        LD (BAS+PTR),DE
        CALL RSTREG
        RET
```

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
RSTBAS:
;RESETS BAS TO PREVIOUS VALUE
        CALL SAVREG
        LD HL,(BAS+PTR)
        LD (TOP+PTR),HL
        LD DE,BAS+PTR
        LDI
        LDI
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
DEFLOC:
INH.4:
;PUSHES A NEW LOCAL INDEX ONTO STACK
        CALL SAVREG
        LD DE,(TOP+PTR)
        CALL GETNOD     ;GET A NEW NODE INDES
        LD HL,IDX+HDR   ;(HDR+IDX) HOLDS INDEX
        LDI
        LDI             ;(DE)<-INDEX
        LD (TOP+PTR),DE
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
PSHINH:
        CALL SAVREG
        LD DE,(TOP+PTR)
        LD HL,IDX+HDR
        LDI
        LDI
        LD (TOP+PTR),DE
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
POPINH:
        CALL SAVREG
        LD IX,(TOP+PTR)
        DEC IX
        DEC IX
        LD (TOP+PTR),IX
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;
GETATR:
;UNSTACKS (NINH) INHERITED AND (NSYN)
;SYNTHESIZED ATTRIBUTES FROM INHSTK
;FETCHES THE INHER. ATTR. AND DEFINES DESCRIPTOR
;BLOCKS FOR EACH IN ATRBLK.  STORES SYN. ATTR.
;INDICES IN ATRBLK.  GIVES THE ADDRS OF THE
;DESC. BLKS. TO CALLER IN (DESC)
        CALL SAVREG
        LD IX,ATRLST
        LD IY,(DESC+ATR)
        LD HL,(BAS+PTR)
        INC HL
        INC HL
;HL=INHSTK, IX=LST OF DESC BLKS
;THIS LOOP UNSTACKS THE INHER. ATTR.
        LD BC,(NINH+ATR)
        CALL STLUP1
GA.1:   CALL LUP1
        JP M,GA.2
        LD DE,IDX+HDR
        LDI
        LDI
;FETCH THE INHER. ATTR.
        CALL FETCH
;COPY HDR TO ATRBLK
        LD (TEMP),HL
        LD HL,IDX+HDR
        LD E,(IX)
        LD D,(IX+1)
;COPY TO CALLERS LOCAL LIST
        LD (IY),E
        LD (IY+1),D
;COPY ENTIRE HDR BLK TO ATRBLK
        LD BC,11
        LDIR
        LD HL,(TEMP)
        LD BC,2
        ADD IX,BC
        ADD IY,BC
        JP GA.1
;
GA.2:
;THIS LOOP JUST STORES THE SYN ATTR INDICES
        LD BC,(NSYN+ATR)
        CALL STLUP1
GA.3:   CALL LUP1
        JP M,GA.4
        LD E,(IX)
        LD D,(IX+1)
;COPY TO CALLERS LOCAL LIST
        LD (IY),E
        LD (IY+1),D
```

```
;COPY INDEX TO ATRBLK
        LDI
        LDI
        LD BC,2
        ADD IX,BC
        ADD IY,BC
        JP GA.3
GA.4:
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ALOSYN:
;ALLOCATES THE INHERITED ATTR., USING
;IDX, TYP AND SPC IN ATRBLK
        CALL SAVREG
;MOVE UP TO FIRST SYN. ATTR. IN ATRLST
        LD IX,ATRLST
        LD BC,(NINH+ATR)
        ADD IX,BC
        ADD IX,BC
        LD BC,(NSYN+ATR)
        CALL STLUP1
AS.1:   CALL LUP1
        JP M,AS.2
        LD L,(IX)
        LD H,(IX+1)
;COPY IDX,TYP,SPC TO HDR BLK
        LD DE,IDX+HDR
        LD BC,5
        LDIR
;ALLOCATE THE NODE WITH THIS INFO
        CALL ALLOC
;NOW COPY ADR,FST,LST INTO ATRBLK
        EX DE,HL
        LD BC,6
        LDIR
;REPEAT FOR OTHER SYN ATTR.
        INC IX
        INC IX
        JP AS.1
AS.2:   CALL RSTREG
        RET
;
;***********************************
;
QUES:                            ; CHANGED TO SUPPORT FORTH
;
; FETCHES VALUE OF A BOOLEAN NODE, B
; DISINHERITS B FROM STACK
; RETURNS A ONE ON THE FORTH STACK IF B=T
; RETURNS A ZERO ON THE FORTH STACK IF B=FALSE
```

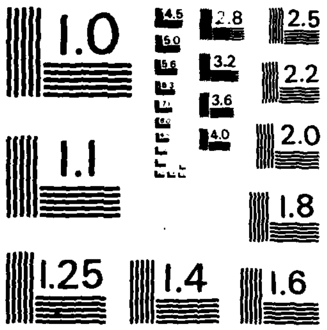
```
        POP     DE                  ; SAVE RETURNS ADDRESS
        CALL    GETX                ;
        CALL    CHKBUL              ; ENSURE BOOLEAN VALUE
        CALL    INH.3
        LD      IX,(FST+XX)         ;
        LD      A,(IX)
        CP      FALSE               ; CHECK VALUE
        JP      NZ,QS1.1            ; => B IS TRUE
        LD      BC,0                ; 0 IS FALSE IN FORTH
        JP      QS1.2
QS1.1:  LD      BC,1                ; 1 IS TRUE IN FORTH
QS1.2:  PUSH    BC                  ; ANSWER FROM QUES1
        PUSH    DE                  ; RESTORE RETURN ADDRESS
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKBUL:
;CHECKS THAT XX IS A BOOLN NODE
        LD A,(TYP+XX)
        CP BOOLN
        RET Z
        LD DE,$CHKBL
        CALL PRLINE
        RET
;
$CHKBL: DB 'TRYING "QUES" ON NON-BOOLN $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GETX:
        CALL SAVREG
        LD BC,1
        LD (NINH+ATR),BC
        LD BC,0
        LD (NSYN+ATR),BC
        LD BC,XATR
        LD (DESC+ATR),BC
        CALL GETATR
        LD DE,XX
        LD HL,(XATR)
        LD BC,11
        LDIR
        CALL RSTREG
        RET
;
TEMP:   DW      0
XATR:   DW
XX:     DS      12
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ATR:    DS      6
```

```
ATRLST: DW      ATRBLK
        DW      ATRBLK+1*BLKSIZ
        DW      ATRBLK+2*BLKSIZ
        DW      ATRBLK+3*BLKSIZ
        DW      ATRBLK+4*BLKSIZ
        DW      ATRBLK+5*BLKSIZ
        DW      ATRBLK+6*BLKSIZ
        DW      ATRBLK+7*BLKSIZ
        DW      ATRBLK+8*BLKSIZ
        DW      ATRBLK+9*BLKSIZ
ATRBLK: DS      10*BLKSIZ
;
        END
;
;END OF ATRB AND COND
;.....................................................
```

```
;
; 31 JUL 83 - ORIGINAL
; 25 OCT 83 - CHANGED ASCII TO DS
; 14 DEC 83 - REMOVED ALL EXTRANEOUS STORAGE, ADDED ZERO O
;
        TITLE BLCK A/O 14 DEC 83
;
;THE BLCK OF STORAGE AREAS FOR BADJR
;
GLOBAL INITSTOR
GLOBAL PTR,HDR,NODLST,NODES,INHSTK
GLOBAL SAVREG,RSTREG,FORSAV,FORRST
GLOBAL STLUP1,LUP1,STLUP2,LUP2,STLUP3,LUP3
;
                        .XLIST
        MACLIB EQATMO
        EQUATES
                        .LIST
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
INITSTOR:
;INITIALIZES NODE LIST, NODE SPACE PTRS,
;INHSTK PTRS, AND STACK
; !!!!!!!!!!!!! NOTE !!!!!!!!!!!
;CALL INITSTOR ONLY ONCE !!!!!!!
;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
;SET STACK PTR FOR SAVING REGISTERS
;
        LD HL,REGSTK
        LD (REGTOS),HL
        CALL SAVREG
        LD HL,NODES
        LD (BASE+PTR),HL
        LD (FREE+PTR),HL
        LD BC,MAXSTOR
        ADD HL,BC
        LD (LAST+PTR),HL
;
; ZERO OUT INHSTK
;
        LD      HL,INHSTK
        LD      DE,INHSTK
        INC     DE
        LD      (HL),0
        LD      BC,200H
        DEC     BC
        LDIR
;
; ZERO OUT STRING SPACE
;
        LD HL,NODES
        LD DE,NODES
        INC DE
        LD (HL),0
        LD BC,MAXSTOR
        DEC BC
        LDIR
;MARK ALL NODES AVAIL IN NODLST
        LD HL,NODLST
        LD (HL),NILIDX
        INC HL
        LD (HL),NILIDX
        INC HL
        LD (HL),0
        INC HL
        LD (HL),0
        INC HL
        EX DE,HL
        LD HL,NODLST
        LD BC,NUMNOD
        DEC BC
; MULTIPLY BC BY 4
        SLA C
        RL B
        SLA C
        RL B
        LDIR
;
;INITIALIZE POINTERS TO INHSTK
IS.2:   LD HL,INHSTK
        LD (BAS+PTR),HL
        INC HL
        INC HL
        LD (TOP+PTR),HL
        CALL RSTREG
        RET
;
;
; ROUTINES TO SAVE FORTH REGISTERS
;
FORSAV:
        LD      (BCSAV),BC
        LD      (DESAV),DE
        LD      (HLSAV),HL
        LD      (IXSAV),IX
        LD      (IYSAV),IY
        RET
;
FORRST:
        LD      IY,(IYSAV)
        LD      IX,(IXSAV)
        LD      HL,(HLSAV)
        LD      DE,(DESAV)
        LD      BC,(BCSAV)
        RET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SAVREG:
;SAVES ALL REGISTERS HERE
        LD (TEMP),SP
        LD SP,(REGTOS)
        PUSH BC
        PUSH DE
        PUSH HL
        PUSH IX
        PUSH IY
        LD (REGTOS),SP
        LD SP,(TEMP)
        RET
;
RSTREG:
;RESTORES ALL REGISTERS FROM REGBLK
        LD (TEMP),SP
        LD SP,(REGTOS)
        POP IY
        POP IX
        POP HL
        POP DE
        POP BC
        LD (REGTOS),SP
        LD SP,(TEMP)
        RET
;
;;;;;;;;;;;;;;;;;;;; ;;;;;;;;
;
;SET A DOLOOP COUNTER TO BC
STLUP1:
        LD (CLUP1),BC
        RET
;
STLUP2:
        LD (CLUP2),BC
        RET
;
STLUP3:
        LD (CLUP3),BC
        RET
;
;EACH OF THESE DECREMENTS LOOP COUNTER BY 1
;
LUP1:
        LD (TEMP),BC
        LD BC,(CLUP1)
        DEC C
        JP P,LUP11
        DEC B
LUP11:  LD (CLUP1),BC
        LD BC,(TEMP)
        RET
;
LUP2:
        LD (TEMP),BC
        LD BC,(CLUP2)
        DEC C
        JP P,LUP22
        DEC B
LUP22:  LD (CLUP2),BC
        LD BC,(TEMP)
        RET
;
LUP3:
        LD (TEMP),BC
        LD BC,(CLUP3)
        DEC C
        JP P,LUP33
        DEC B
LUP33:  LD (CLUP3),BC
        LD BC,(TEMP)
        RET
;
CLUP1:  DW      0
CLUP2:  DW      0
CLUP3:  DW      0
;
;*****************************
; DATA STORAGE AREA
;*****************************
;
        DS      100H
STACK:  DS       80H
;
TEMP:   DW      0
REGTOS: DW      REGSTK
        DS      100H
REGSTK: DW      0
BCSAV:  DW      1
DESAV:  DW      1
HLSAV:  DW      1
IXSAV:  DW      1
;
PTR:    DS      20H
HDR:    DS      20H
INHSTK: DS      200H
NODLST: DS      4*NUMNOD
NODES:  DS      MAXSTOR
IYSAV:  DW      1
; END OF BLCK...................
        END
```

```
AD-A139 424   FORJR: AN IMPLEMENTATION OF BADJR USING FORTH AND Z80     2/2
              ASSEMBLY LANGUAGE(U) AIR FORCE INST OF TECH
              WRIGHT-PATTERSON AFB OH  W M EDMONSON 1983
UNCLASSIFIED  AFIT/CI/NR-83-87T                          F/G 9/2       NL
```



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
;
;  6 OCT 83
; 14 DEC 83 - REMOVED STREAM? AND DRY?
; 21 DEC 83 - MODIFIED THE BOOLEAN OPERATORS
; 06 JAN 84 - CORRECTED BXOR
; 25 JAN 84 - CORRECTED BOOLEAN?
;
        TITLE BOOL A/O 25 JAN 84
;
GLOBAL BAND,BOR,BXOR,BNOT
GLOBAL ATOM?,NIL?,SYMBOL?,NUMBER?,BOOLEAN?
GLOBAL EMPTY?,SEQUENCE?
;
EXTERNAL GETATR,ALOSYN,ATR,PRLINE
EXTERNAL SAVREG,RSTREG
;
        .XLIST
        MACLIB EQATMO
        EQUATES
                .LIST
;
;LOGICAL FUNCTIONS 'AND,OR,XOR' TAKE TWO
;ATTRIBUTES X,Y AND COMPUTE RESULT Z
;'NOT' TAKES 1 ATTR. AND PRODUCES ITS COMPLEMENT
;PREDICATES 'ATM?,SEQ?,STR?' TAKE 1 ATTR. X
;AND PRODUCE 1 LOGICAL ATTR Z
;
;FIRST 3 LOGICAL FUNCTIONS USE GT2BOL TO
;FETCH X,Y, GET INDEX OF Z, RETURN A=X+Y
;WHERE T=54H, F=46H
;
BAND:
        CALL GT2BOL
        CP      0A8H            ; TRUE + TRUE
        JP NZ,STFAL
        JP STTRU
;
BOR:
        CALL GT2BOL
        CP      09AH            ; TRUE + FALSE
        JP M,STFAL
        JP STTRU
;
BXOR:
        CALL GT2BOL
        CP      09AH            ; TRUE + FALSE
        JP Z,STTRU              ; 6 JAN
        JP STFAL                ; 6 JAN
;
BNOT:
        CALL GT1BOL
        CP      FALSE           ; FALSE?
        JP NZ,STFAL
        JP STTRU
;
STTRU:  LD A,TRUE
        JP STCOM
;
STFAL:  LD A,FALSE
STCOM:
        LD (BRES),A
        CALL STBOOL
        RET
;
GT1BOL:
        LD BC,1
        LD (NINH+ATR),BC
        LD (NSYN+ATR),BC
        LD BC,YATR
        LD (DESC+ATR),BC
        CALL GETATR
GT1B.1:
        LD HL,(YATR)
        LD DE,YY
        LD BC,11
        LDIR
        LD HL,(ZATR)
        LD DE,ZZ
        LDI
        LDI
        LD A,(TYP+YY)
        CALL CHKBOL
        LD IY,(YY+FST)
        LD A,(IY)
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GT2BOL:
;GET 2 ATTR. FROM STACK, CHECK IF BOOLEAN
;MAKE A=X+Y (T=1,F=0), SAVE INIYX OF RESULT
        LD BC,2
        LD (NINH+ATR),BC
        LD BC,1
        LD (NSYN+ATR),BC
        LD BC,XATR
        LD (DESC+ATR),BC
        CALL GETATR
        LD HL,(XATR)
        LD DE,XX
        LD BC,11
        LDIR
        LD A,(TYP+XX)
        CALL CHKBOL
        CALL GT1B.1
        LD IX,(FST+XX)
        ADD A,(IX)
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
STBOOL:
;SAVES VALUE OF A AS LOG. RESULT
;ALLOCATES A BOOLEAN NODE, STORES RESULT
        LD A,BOOLN
        LD (TYP+ZZ),A
        LD BC,1
        LD (ZZ+SPC),BC
        LD DE,(ZATR)
        LD HL,ZZ
        LD BC,5
        LDIR
        CALL ALOSYN
        EX DE,HL
        LD BC,6
        LDIR
        LD IX,(FST+ZZ)
        LD A,(BRES)
        LD (IX),A
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ATOM?:
        CALL GTTYP
        CP ATOM
        JP M,STTRU
        JP STFAL
;
NIL?:
        CALL GTTYP
        CP STREM
        JP Z,STFAL
        CP NIL
        JP Z,STTRU
        CALL CHKMT
        JP Z,STTRU
        JP STFAL
;
SYMBOL?:
        CALL GTTYP
        AND OFOH
        CP SYMBL
        JP Z,STTRU
        JP STFAL
;
NUMBER?:
        CALL GTTYP
        AND OFOH
        CP NUMBR
        JP Z,STTRU
        JP STFAL
;
BOOLEAN?:
        CALL GTTYP
        CP BOOLN
        JP Z,BOOL1              ; 1/25
        JP STFAL
BOOL1:  LD      HL,6+YY ; 1/25 GET 1ST CHAR OF SYMBOL
        LD      B,(HL)
        INC     HL
        LD      C,(HL)
        LD      A,(BC)
        CP      TRUE            ; IS TRUE?
        JP      Z,STTRU         ;    YES - SYMBOL IS BOOLE
        CP      FALSE           ; IS FALSE?
        JP      Z,STTRU         ;    YES - SYMBOL IS BOOLE
        JP      STFAL           ;    NO  - SYMBOL IS NOT B
;
EMPTY?:
        CALL GTTYP
        AND OFOH
        CP SYMBL
        JP NZ,STFAL
        CALL CHKMT
        JP Z,STTRU
        JP STFAL
;
SEQUENCE?:
        CALL GTTYP
        CP SEQNC
        JP Z,STTRU
        JP STFAL
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GTTYP:
        CALL GTTWO
        LD A,(TYP+YY)
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GTTWO:
        LD BC,1
        LD (NINH+ATR),BC
        LD (NSYN+ATR),BC
        LD BC,YATR
        LD (DESC+ATR),BC
        CALL GETATR
        LD HL,(YATR)
        LD DE,YY
        LD BC,11
        LDIR
        LD HL,(ZATR)
        LD DE,ZZ
        LDI
        LDI
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKBOL:
;CHECKS THAT A=(TYP) IS 'BOOLN'
        CP BOOLN
        RET Z
        LD DE,$CHKB             ; TYPE IS NOT BOOLEAN
        CALL PRLINE
        RET
;
$CHKB: DB 'TRYING LOGICAL OP ON NON-BOOLN $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKMT:
        LD HL,(SPC+YY)
        LD DE,0
        OR 0
        SBC HL,DE
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
XATR:   DW      0
YATR:   DW      0
ZATR:   DW      0
XX:     DS      12
YY:     DS      12
ZZ:     DS      12
BRES:   DB      0
        END
;END OF BOOL........................................
```

```
;
;  6 OCT 83
; 15 DEC 83 - REMOVED DEAD WOOD STORAGE AND MODULES
; 13 JAN 84 - REMOVED TR FROM CONVERSION MODULES
;
        TITLE CONV A/O 13 JAN 84
;
;
GLOBAL ID,SYMSEQ
GLOBAL SEQSYM,SEQNUM,NUMSYM,RV
GLOBAL DL,DR,SEL,SER
;
EXTERNAL ATR,GETATR,ALOSYN,PRLINE,HDR
EXTERNAL GETNOD,FETCH,ALLOC,SAVREG,RSTREG
EXTERNAL BCDHEX,BCDASC,PSHINH,POPINH
EXTERNAL MAKNUM,MAKSYM,INPBUF
EXTERNAL SETINH,STKINH,RSTINH
;
EXTERNAL INH.1,INH.2,INH.3,INH.4

                        .XLIST
;
MACLIB EQATMO
        EQUATES
;
        MACLIB  MACROS
;
                        .LIST
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ID:
;MAKES Z AN IDENTICAL COPY OF X
;UNSTACK X,Y AND DO FETCH ON X
        CALL GETYZ
;ALLOCATE THE Y NODE
ID.1:
        LD DE,(SPC+YY)
        LD (SPC+ZZ),DE
        LD A,(TYP+YY)
        LD (TYP+ZZ),A
        CALL ALOCZ
;NOW COPY DATA OF X -> Y
;UNLESS X IS NIL, EMPTY OR DRY
        CALL CHKMT
        RET Z
        LD DE,(FST+ZZ)
        LD HL,(FST+YY)
        LD BC,(SPC+YY)
        LDIR
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;
SYMSEQ:
;CONVERTS SYMBOL X TO SEQ OF INDIVIDUAL CHARS Y
        CALL GETYZ
;CHECK THAT X IS SYMBL
        CALL CHKSYM
        JP NZ,SMSQ.9
        LD A,SEQNC
        LD (TYP+ZZ),A
;CHECK IF X IS EMPTY
        CALL CHKMT
        JP Z,MAKNIL
;X=SYMBL, SO ALLOCATE Y. SPC(Y)=2*SPC(X)
;SINCE EACH CHAR REQUIRES ITS OWN NODE
        LD BC,(SPC+YY)
        SLA C
        RL B
        LD (SPC+ZZ),BC
;ALLOC Y
        CALL ALOCZ
        LD DE,(FST+ZZ)
;SPC=1, TYP=SYMBL FOR ALL NEW NODES
        LD BC,1
        LD (SPC+HDR),BC
        LD A,SYMBL
        LD (TYP+HDR),A
;LOOP TO MAKE AND STORE NODES FOR EACH CHAR IN X
        LD BC,(SPC+YY)
        CALL STLUP1
        LD IX,(FST+YY)
SMSQ.1: CALL LUP1
        JP M,SMSQ.2
;GET A NEW NODE
        CALL GETNOD
;STORE ITS INDEX IN Y
        LD HL,IDX+HDR
        LDI
        LDI
;ALLOC THE NEW NODE (1-BYTE LONG)
;AND STORE NEXT BYTE FROM X
        CALL ALLOC
        LD IY,(FST+HDR)
;GET NEXT CHAR IN X
        LD A,(IX)
        INC IX
        LD (IY),A
        JP SMSQ.1
SMSQ.2: RET
;
SMSQ.9: LD DE,$SMSQ
        JP QUIT
$SMSQ:  DB 'TRYING SYMSEQ ON NON-SYMBL $'
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;SEQSYM AND SEQNUM USE A ROUTINE SQSYNM
;WHICH DOES MOST OF THE WORK
SEQSYM:
        CALL GETYZ
;CHECK THAT X IS A SEQNC
        CALL CHKSEQ
        JP NZ,SQSM9
        LD A,SYMBL
        LD (TYP+ZZ),A
        CALL CHKMT
        JP Z,MAKNIL
        CALL SQSYNM
;PASS IDX(Y) TO MAKSYM
        LD DE,(IDX+ZZ)
        LD (IDX+HDR),DE
;CALL MAKSYM TO STORE CONTENTS OF INPBUF IN Y
        CALL MAKSYM
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SEQNUM:
;SQSYNM FILLS INPBUF WITH ASCII DIGITS OF X
        CALL GETYZ
;CHECK THAT X IS A SEQNC
        CALL CHKSEQ
        JP NZ,SQSM9
        LD A,NUMBR
        LD (ZZ+TYP),A
        CALL CHKMT
        JP Z,MAKNIL
        CALL SQSYNM
;GIVE INDEX OF Y TO MAKNUM, WHICH DOES
;EVERYTHING ELSE
        LD DE,(IDX+ZZ)
        LD (IDX+HDR),DE
        CALL MAKNUM
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SQSYNM:
;CONCATENATES A SEQ OF SYMBOLS (X) INTO
;ONE SYMBOL IN INPBUF
;MAKNUM OR MAKSYM CONVERT THIS SYMBOL
;INTO A NUMBER OR SYMBOL NODE
;ADD UP ALL THE LENGTHS OF THE SYMBOLS IN X
;CONCATENATE SYMBOLS INTO INPBUF
;ALSO CHECK THAT ALL ARE SYMBL'S
        LD BC,(SPC+YY)
        LD IX,(FST+YY)
```

```
;DIVIDE BC BY 2
        SRL B
        RR C
        CALL STLUP1
;SET TOTAL COUNT TO ZERO
        LD IY,0
;SET DE TO INPBUF
        LD DE,INPBUF
SQSM1:  CALL LUP1
        JP M,SQSM2
;FETCH NEXT OBJ IN X
        CALL NXTOBJ
;MAKE SURE IT IS A SYMBOL
        LD A,(TYP+XNXT)
        CP SYMBL
        JP NZ,SQSM91
;ADD SPC(IX) TO IY
        LD BC,(SPC+XNXT)
        ADD IY,BC
;COPY BC BYTES INTO INPBUF
        LD HL,(FST+XNXT)
        LDIR
        JP SQSM1
SQSM2:  LD (TEMP),IY
        LD A,(TEMP)
;NOTE: ONLY 8 LSB OF COUNT ARE INCLUDED!
        LD (INPBUF-1),A
        RET
;
SQSM9:  LD DE,$SQSM
        JP QUIT
$SQSM:  DB 'TRYING SEQSYM OR SEQNUM ON NON-SEQ $'
SQSM91: LD DE,$SQSM1
        JP QUIT
$SQSM1: DB 'TRYING TO INCLUDE NON-SYMBL IN INPBUF $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NUMSYM:
;CONVERTS A PACKED BCD NUMBER IN X
;TO ASCII DIGITS (PLUS '+','-','.') IN Y
        CALL GETYZ
;CHECK THAT X IS A NUMBER
        CALL CHKNUM
        JP NZ,NMSY.9
        LD A,SYMBL
        LD (TYP+ZZ),A
        CALL CHKMT
        JP Z,MAKNIL
;USE INPBUF(DE) TO SEND DIGITS TO MAKSYM
        LD DE,INPBUF
;SET CHAR COUNT TO 2, 1 EACH FOR '+/-' AND '.'
        LD IY,2
```

```
;MAKE FST CHAR A '+' OR '-'
        LD A,(TYP+YY)
        CP POSFXP
        LD A,'+'
        JP Z,NMSY.1
        LD A,'-'
NMSY.1: LD (DE),A
        INC DE
;IX PTS TO WHOLE/FRAC BYTE COUNTS
;HL TO FIRST DIGIT
        LD IX,(FST+YY)
        LD HL,(FST+YY)
        INC HL
        INC HL
;CONVERT THE WHOLE DIGITS TO CHARS
        LD C,(IX)
        LD A,C
        CP 0
        JP Z,NMSY.2
        LD B,0
;FOR BCDASC: DE=DEST.,HL=SOURCE,BC=#BCD
        CALL BCDASC
;ADD BC TO HL , BC*2 TO DE AND IY
        ADD HL,BC
        SLA C
        RL B
        EX DE,HL
        ADD HL,BC
        ADD IY,BC
        EX DE,HL
NMSY.2:
;STORE A '.'
        LD A,'.'
        LD (DE),A
        INC DE
;NOW CONVERT THE FRACTION DIGITS
        LD C,(IX+1)
        LD A,C
        CP 0
        JP Z,NMSY.3
        LD B,0
        CALL BCDASC
        SLA C
        RL B
        ADD IY,BC
NMSY.3:
;SET TOTAL CHAR COUNT (255 MAX)
        LD (TEMP),IY
        LD A,(TEMP)
        LD (INPBUF-1),A
;GIVE IDX(Y) TO MAKSYM
        LD DE,(IDX+ZZ)
        LD (IDX+HDR),DE
        CALL MAKSYM
        RET
;
NMSY.9: LD DE,$NMSY
        JP QUIT
$NMSY:  DB 'TRYING NUMSYM ON NON-NUMBER $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
RV:
;COPIES SEQ X TO Y, OBJECTS IN REVERSE ORDER
        CALL GETYZ
;CHECK THAT X IS SEQNC
        CALL CHKSEQ
        JP NZ,RV.9
        LD A,SEQNC
        LD (TYP+ZZ),A
        CALL CHKMT
        JP Z,MAKNIL
;ALLOCATE Y, SAME SIZE AS X
        LD HL,(SPC+YY)
        LD (SPC+ZZ),HL
        CALL ALOCZ
;COPY INDICES OF X TO Y, IN REVERSE
        LD BC,(SPC+YY)
;BC=# OF INDICES
        SRL B
        RR C
        CALL STLUP1
        LD IX,(FST+YY)
        LD IY,(LST+ZZ)
        DEC IY
RV.1:   CALL LUP1
        JP M,RV.2
        LD E,(IX)
        LD D,(IX+1)
        LD (IY),E
        LD (IY+1),D
        INC IX
        INC IX
        DEC IY
        DEC IY
        JP RV.1
RV.2:   RET
;
RV.9:   LD DE,$RV
        JP QUIT
$RV:    DB 'TRYING RV ON NON-SEQNC $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
DL:
;DISTRIBUTE LEFT, FLAGGED BY 'L'
```

```
        LD A,'L'
        JP DLR.0
DR:
;DISTRIBUTE RIGHT, FLAGGED BY 'R'
        LD A,'R'
DLR.0:  LD (L.OR.R),A
;UNSTACK X,Y,Z
        CALL GETXYZ
;CHECK THAT X IS SEQNC
        LD A,(TYP+XX)
        CP SEQNC
        JP NZ,DLR.91
;ALLOCATE Z SAME LN AS X
        LD A,SEQNC
        LD (TYP+ZZ),A
        CALL CHKMT
        JP Z,MAKNIL
        LD BC,(SPC+XX)
        LD (SPC+ZZ),BC
        CALL ALOCZ
;IX,DE PT TO X,Z DATA SPACE
        LD IX,(FST+XX)
        LD DE,(FST+ZZ)
;SET UP DO-LOOP TO STEP THRU X
        LD BC,(SPC+XX)
        SRL B
        RR C
        CALL STLUP1
DLR.1:  CALL LUP1
        JP M,DLR.2
;GET A NEW NODE
        CALL GETNOD
;STORE ITS INDEX IN Z (DE)
        LD HL,IDX+HDR
        LDI
        LDI
;SET TYP,SPC FOR NEW NODES IN Z
        LD A,SEQNC
        LD (TYP+HDR),A
        LD BC,4
        LD (SPC+HDR),BC
;ALLOCATE THE NEW NODE
        CALL ALLOC
;SAVE ITS FST ADDR IN IY
        LD IY,(FST+HDR)
;GET NXT OBJECT OF X
        CALL NXTOBJ
;PUT ITS INDEX IN HL
        LD HL,(IDX+HDR)
;STORE IN NEW NODE, EITHER
;<Y,X.I>(DL) OR <X.I,Y>(DR)
        LD A,(L.OR.R)
        CP 'R'
        JP Z,DROP
;IT'S DL
DLOP:   LD BC,(IDX+YY)
        LD (IY),C
        LD (IY+1),B
        LD (IY+2),L
        LD (IY+3),H
        JP DLR.1
;OR, IT'S DIR
DROP:   LD BC,(IDX+YY)
        LD (IY),L
        LD (IY+1),H
        LD (IY+2),C
        LD (IY+3),B
        JP DLR.1
;ALL DONE
DLR.2:  RET
;ERROR MSG
DLR.91: LD DE,$DLR91
        JP QUIT
$DLR91: DB 'TRYING DL/R ON NON-SEQNC $'
;
L.OR.R: DW      0
;
;SELECT LEFT/RIGHT
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SEL:
;SEL FLAGGED BY 'L'
        LD A,'L'
        JP SL.0
SER:
;SER FLAGGED BY 'R'
        LD A,'R'
SL.0:   LD (L.OR.R),A
;UNSTACK X,Y;Z
        CALL GETXYZ
;CHECK THAT X IS NON-NIL SEQNC
        LD A,(TYP+XX)
        CP SEQNC
        JP NZ,SL.91
        LD A,SEQNC
        LD (TYP+ZZ),A
        CALL CHKMT
        JP Z,MAKNIL
;CHECK THAT Y IS A LEGAL #
SL.1:   LD A,(TYP+YY)
        CP POSFXP
        JP NZ,SL.92
;INTEGER PART MUST BE < 1.E6 (ARBITRARY)
        LD IY,(FST+YY)
        LD A,(IY)
        CP 0
        JP Z,SL.92
        CP 4
        JP P,SL.92
;CONVERT Y (HL) TO HEX IN (YHEX)
        LD C,(IY)
        LD HL,(FST+YY)
        INC HL
        INC HL
        LD DE,YHEX
        CALL BCDHEX
;COMPARE Y TO 0 AND LN(X)
        LD BC,0
        LD HL,(YHEX)
        OR 0
        SBC HL,BC
        JP Z,SL.92
;HL=# OF OBJECTS IN X
        LD HL,(SPC+XX)
        SRL H
        RR L
        OR 0
;SUB Y TO SEE IF Y># OF OBJS
        LD BC,(YHEX)
        SBC HL,BC
        JP M,SL.92
;Y IS OK, SO IS X
;BC=LN(X), LET DE=2*Y-2
SL.2:   LD DE,(YHEX)
        SLA E
        RL D
        DEC DE
        DEC DE
        XOR A
;DO EITHER SEL OR SER
        LD A,(L.OR.R)
        CP 'R'
        JP Z,SEROP
SELOP:  LD HL,(FST+XX)
        ADD HL,DE
        JP SL.3
;
SEROP:  LD HL,(LST+XX)
        DEC HL
        XOR A
        SBC HL,DE
;HL NOW PTS TO SELECTED OBJ IN X
;SET UP INHSTK SO ID CAN JUST COPY X.Y->Z
SL.3:   LD DE,IDX+HDR
        LDI
        LDI
;PUSH X.Y ONTO INHSTK
        CALL PSHINH
```

```
; SET UP FOR CALL TO ID (MODIFIED 11/22)
        CALL    SETINH          ; 11/22
        LD      BC,4            ; 11/22
        PUSH    BC              ; 11/22
        CALL    STKINH          ; 11/22
        LD      BC,3            ; 11/22
        PUSH    BC              ; 11/22
        CALL    STKINH          ; 11/22
        CALL    ID              ; 11/22
        CALL    RSTINH          ; 11/22
        CALL POPINH
;ALL DONE
        RET
;
SL.91:  LD DE,$SLR91
        JP QUIT
SL.92:  LD DE,$SLR92
        JP QUIT
;
$SLR91: DB 'TRYING SEL/R ON NIL OR NON-SEQNC $'
$SLR92: DB 'FOR X.K: K>!X!, K>1.E6, OR K =< 0 $'
;
YHEX:   DW      0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
QUIT:   CALL PRLINE
        RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
GETXYZ:
;GETS 3 INDICES OFF OF INHSTK
;FETCHES FIRST 2 SO ALL THEIR PROPERTIES ARE KNOWN
        LD BC,2
        LD (NINH+ATR),BC
        LD BC,1
        LD (NSYN+ATR),BC
        LD BC,XATR
        CALL GTYZ.1
        LD DE,XX
        LD HL,(XATR)
        LD BC,11
        LDIR
        RET
;
GETYZ:
;GETS 2 INDICES OFF OF INHSTK
;FETCHES FIRST ONE SO ALL ITS PROPERTIES ARE KNOWN
        LD BC,1
        LD (NINH+ATR),BC
        LD (NSYN+ATR),BC
        LD BC,YATR
GTYZ.1:
        LD (DESC+ATR),BC
        CALL GETATR
;
        LD HL,(YATR)
        LD DE,YY
        LD BC,11
        LDIR
;
        LD HL,(ZATR)
        LD DE,ZZ
        LDI
        LDI
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ALOCZ:
;ALLOCATES A NODE, USING PARAMETERS IN ZZ
        CALL SAVREG
        LD HL,IDX+ZZ
        LD DE,IDX+HDR
        LD BC,5
        LDIR
        CALL ALLOC
;
        EX DE,HL
        LD BC,6
        LDIR
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
MAKNIL:
;MAKES A NODE OF ZERO LENGTH, (TYP+ZZ)
        LD BC,0
        LD (SPC+ZZ),BC
        CALL ALOCZ
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NXTOBJ:
;FETCHES OBJECT WITH INDEX POINTED TO BY IX
;STORES HDR BLCK IN XNXT
        CALL SAVREG
        LD E,(IX)
        LD D,(IX+1)
        LD (IDX+HDR),DE
        CALL FETCH
        LD HL,IDX+HDR
        LD DE,XNXT
        LD BC,11
        LDIR
        CALL RSTREG
        INC IX
        INC IX
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
STLUP1:
        LD (CLUP1),BC
        RET
LUP1:
        LD (BCTEMP),BC
        LD BC,(CLUP1)
        DEC C
        JP P,DL.1
        DEC B
DL.1:   LD (CLUP1),BC
        LD BC,(BCTEMP)
        RET
CLUP1:  DW      0
STLUP2:
        LD (CLUP2),BC
        RET
LUP2:
        LD (BCTEMP),BC
        LD BC,(CLUP2)
        DEC C
        JP P,DL.2
        DEC B
DL.2:   LD (CLUP2),BC
        LD BC,(BCTEMP)
        RET
CLUP2:  DW      0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKSYM:
        LD A,(TYP+YY)
        CP SYMBL
        RET
;
CHKNUM:
        LD A,(TYP+YY)
        AND 0F0H
        CP NUMBR
        RET
CHKSEQ:
        LD A,(TYP+YY)
        CP SEQNC
        RET
CHKMT:
;RETURNS Z IF (SPC+YY)=0
        CALL SAVREG
        LD HL,(SPC+YY)
        LD BC,0
        OR 0
        SBC HL,BC
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
TEMP:   DW      0
BCTEMP: DW      0
XATR:   DW      0
YATR:   DW      0
ZATR:   DW      0
XX:     DS      12
YY:     DS      12
ZZ:     DS      12
XNXT:   DS      12
;
        END
;.................................................
```

```
;
;6 OCT 83 - REMOVED SYNTAX ERRORS
;27 OCT 83 - CHANGED SYMIMM & NUMIMM
;14 NOV 83 - CHANGED SELIMM
; 21 DEC 83 - REMOVED DEADWOOD MODULES AND STORAGE AREAS
;
        TITLE IMED A/O 21 DEC 83
;
;
GLOBAL NUMIMM,SYMIMM,SELIMM,LENIMM
GLOBAL MERIMM
GLOBAL CONIMM
EXTERNAL ALLOC,FETCH,GETNOD,SAVREG,RSTREG
EXTERNAL MAKNUM,MAKSYM,PRLINE,HEXBCD
EXTERNAL PSHINH,POPINH,PTR,HDR,INPBUF
EXTERNAL RSTBAS
;
                        .XLIST
        MACLIB EQATMO
        EQUATES
;
                        .LIST
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NUMIMM:
;COPY ASCII DIGITS TO INPBUF
; GET # OF CHARS IN NUMBER
        POP     DE              ; SAVE RETURN ADDRESS
        POP     BC              ; GET # OF CHARS
        LD      (BCTEMP),BC
        POP     HL
        LD      (HLTEMP),HL
        PUSH    DE              ; RESTORE RETURN ADDRESS
        XOR     A               ; CHECK FOR ZERO SYMBOLS
        ADD     A,C
        JP      Z,NI.92         ; NO DIGITS ENTERED
; GET ADDR OF FIRST CHAR
NI.1:   DEC     C
        JP      M,NI.2          ; ALL CHARACTERS PROCESSED
        LD      A,(HL)
        INC     HL
        CP '+'
        JP Z,NI.1
        CP '-'
        JP Z,NI.1
        CP '.'
        JP Z,NI.1
        CP ' '
        JP Z,NI.1
        CP 03AH
        JP P,NI.9
        CP 30H
        JP M,NI.9
        JP NI.1
NI.2:   LD IX,INPBUF-1
        LD      BC,(BCTEMP)
        LD (IX),C
        LD DE,INPBUF
        LD      HL,(HLTEMP)
        LDIR
;GET A NODE INDEX
        CALL GETNOD
;PUT THE INDEX ON THE STACK
        CALL PSHINH
        CALL MAKNUM
        RET
;
NI.9:   LD DE,$NI91
        CALL PRLINE
        RET
NI.92:  LD DE,$NI92
        CALL PRLINE
        RET
$NI91:  DB 'ILLEGAL CHAR. IN IMMED NUM $'
$NI92:  DB 'NIL INPUT ON IMMED NUM $'
;
HLTEMP: DW      0
BCTEMP: DW      0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SYMIMM:
;COPY ASCII CHARS TO INPBUF
; GET # OF CHARS IN SYMBOL
        POP     DE              ; SAVE RETURN ADDRESS
        POP     BC              ; # OF SYMBOLS
        POP     HL              ; ADDR OF FIRST SYMBOL
        PUSH    DE              ; RESTORE RETURN ADDRESS
        XOR     A               ; CLEAR ACCUMULATOR
        SBC     A,B             ; CHECK FOR > 255 SYMBOLS
        JP      M,SI.9
        XOR     A               ; CHECK FOR 0 SYMBOLS
        ADD     A,C
        JP      Z,SI.9
; COPY SYMBOLS INTO BUFFER
        LD IX,INPBUF-1
        LD (IX),C
;SET DE TO INPBUF
        LD DE,INPBUF
;COPY CHARS
        LDIR
;GET A NODE INDEX
        CALL GETNOD
;PUT THE INDEX ON THE STACK
        CALL PSHINH
```

```
;CALL MAKSYM TO ALLOC NODE AND STORE CHARS
        CALL MAKSYM
        RET
SI.9:   LD DE,$SI9
        CALL PRLINE
        RET                         ; 11/14
$SI9:   DB '0 OR >255 CHARS IN IMMED SYM $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SELIMM:
;GETS SEQNC X FROM TOP OF INHSTK, AND
;REPLACES IT WITH X.I
        CALL SAVREG
        POP     DE                  ; 11/14 - SAVE RETURN ADDR
        POP     BC                  ; 11/14 - GET ITH INDEX IN
        PUSH    DE                  ; 11/14 - RESTORE RETURN A
;FETCH TOP-MOST INDEX, PUT IN (IDX+HDR)
        CALL FETTOP
;FETCH FST ADDR OF X
        CALL FETCH
;CHECK THAT X IS SEQNC AND !X!=>I
        LD A,(TYP+HDR)
        CP SEQNC
        JP NZ,IS.10
        LD HL,(SPC+HDR)
        SLA C
        RL  B
        AND 0
        SBC HL,BC
        JP M,IS.10
        LD IY,(FST+HDR)
;BC = 2*I, SET BY SEL MACRO
        DEC BC
        DEC BC
;POINT IY TO I'TH ENTRY IN X
        ADD IY,BC
        LD L,(IY)
        LD H,(IY+1)
        LD (IDXIMM),HL
;POP X
        CALL POPINH
;PUSH X.I
        LD HL,(IDXIMM)
        LD (IDX+HDR),HL
        CALL PSHINH
        CALL RSTREG
        RET
;ERROR MESSAGE
IS.10:  LD DE,$SEL
        CALL PRLINE
        CALL    RSTREG              ; 11/14
        RET                         ; 11/14
```

```
$SEL:   DB 'TRYING SEL ON NON-SEQNC, OR TOO SHORT $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
LENIMM:
;GETS SEQNC X FROM TOP OF INHSTK, AND
;FINDS ITS LENGTH. LEAVES NUMERIC ATOM ON INHSTK.
        CALL SAVREG
;FETCH TOP-MOST INDEX, PUT IN (IDX+HDR)
        CALL FETTOP
        CALL FETCH
;CHECK THAT X IS SEQNC
        LD A,(TYP+HDR)
        CP SEQNC
        JP NZ,IL.10
;CALL HXBC TO CONVERT (SPC+HDR)/2 TO PACKED BCD DIGITS
        LD HL,(SPC+HDR)
        SRL H
        RR L
        LD DE,ILBUF+2
        CALL HEXBCD
;FIND # OF NON-ZERO BYTES
        LD HL,ILBUF+2
        LD BC,2
IL.0:   LD A,(HL)
        CP 0
        JP NZ,IL.1
        INC HL
        DEC C
        JP NZ,IL.0
;SHIFT NON-ZERO DIGITS UP IN ILBUF
IL.1:   INC BC
        LDIR
        EX DE,HL
        LD DE,ILBUF+2
        OR A
        SBC HL,DE
        EX DE,HL
        LD HL,ILBUF
;STORE # OF DIGITS IN ILBUF
        LD (HL),E
        INC DE
        INC DE
;DEFINE NODE SPACE NEEDED
        LD (SPC+HDR),DE
;ALLOCATE NODE OF TYPE POSFXP
        LD A,POSFXP
        LD (TYP+HDR),A
;GET A NEW NODE
        CALL GETNOD
        LD HL,(IDX+HDR)
        LD (IDXIMM),HL
        CALL ALLOC
```

```
;TRANSFER ILBUF TO NODE
        LD DE,(FST+HDR)
        LD BC,(SPC+HDR)
        LD HL,ILBUF
        LDIR
;ALL DONE
;POP X
        CALL POPINH
;PUSH  !X!
        LD HL,(IDXIMM)
        LD (IDX+HDR),HL
        CALL PSHINH
        CALL RSTREG
        RET
;ERROR MESSAGE
IL.10:  LD DE,$LEN
        CALL PRLINE
        CALL    RSTREG
        RET
$LEN:   DB 'TRYING LEN FN ON NON-SEQNC $'
;
ILBUF:  DS      5
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CONIMM:
;UNSTACKS INDICES FROM INHSTK AND MAKES A SEQNC,
;LEAVING SEQNC INDEX ON INHSTK
;
        CALL SAVREG
;COUNT # OF INDICES
        CALL CNTSTK
;GET A NEW INDEX
        CALL GETNOD
        LD DE,(IDX+HDR)
        LD (IDXIMM),DE
;SET NODE TYPE TO SEQNC
        LD A,SEQNC
        LD (TYP+HDR),A
;DEFINE SPACE NEEDED
        LD HL,(CNTIMM)
        SLA L
        RL  H
        LD (SPC+HDR),HL
;ALLOCATE THE NODE
        CALL ALLOC
        LD BC,0
        OR 0
        SBC HL,BC
        JP Z,CON.1
        LD BC,(SPC+HDR)
;TRANSFER THE STACKED INDICES TO THE NODE
        LD HL,(BAS+PTR)
        INC HL
        INC HL
        LD DE,(FST+HDR)
        LDIR
;SET TOP BACK TO BAS PTR
CON.1:
        CALL    RSTBAS
;PUSH NEW NODE INDEX ONTO INHSTK
        LD DE,(IDXIMM)
        LD (IDX+HDR),DE
        CALL PSHINH
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
MERIMM:
;GETS SEQNC'S X,,,Y FROM TOP OF INHSTK
;REPLACES THEM WITH ONE SEQNC Z WITH ALL THEIR
;ELEMENTS.  LEAVES Z ON INHSTK.
        CALL SAVREG
;SET CNTIMM = # OF NODES ON STK
        CALL CNTSTK
;CHECK THAT ALL X,,,Y ARE SEQNC'S
        LD A,SEQNC
        LD (TYPIMM),A
        CALL CHKSTK
;IF A RETURNS .NE. 0, NON-MATCH FOUND
        CP 0
        JP NZ,MI.10
;SET LENSUM = TOTAL LENGTH OF ALL X,,,Y
        CALL TOTSTK
;SPECIFY NODE TYP AND SPC, ALLOCATE
        LD A,(TYPIMM)
        LD (TYP+HDR),A
        LD BC,(LENSUM)
        LD (SPC+HDR),BC
;GET NEW NODE INDEX, ALLOCATE
        CALL GETNOD
        LD HL,(IDX+HDR)
        LD (IDXIMM),HL
        CALL ALLOC
;SET DE = FST ADDR OF NODE, COPY ALL CNTIMM NODES
        LD DE,(FST+HDR)
        LD BC,(CNTIMM)
;IX PTS TO BOTTOM OF CONS STACK
        LD IX,(BAS+PTR)
        INC IX
        INC IX
MI.1:
        DEC C
        JP P,MI.2
        DEC B
        JP M,MI.3
MI.2:   PUSH BC
;USE BC, HL TO GET ELEMENT COUNTS, NODE INDICES
        LD L,(IX)
        LD H,(IX+1)
        INC IX
        INC IX
        LD (IDX+HDR),HL
;FETCH NEXT NODE
        CALL FETCH
        LD BC,(SPC+HDR)
        LD HL,(FST+HDR)
;COPY (SPC+HDR) BYTES FROM NODE
        LDIR
;RESTORE BC=NODE COUNT
        POP BC
;REPEAT LOOP
        JP MI.1
;RESET TOP=CONS+PTR
MI.3:
        CALL RSTBAS
;PUT NEW NODE ON INHSTK
        LD HL,(IDXIMM)
        LD (IDX+HDR),HL
        CALL PSHINH
        CALL RSTREG
        RET
;ERROR MSG
MI.10:  LD DE,$MER
        CALL PRLINE
        CALL    RSTREG
        RET
$MER:   DB 'TRYING MERGE OF NON-SEQNC $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
FETTOP:
;FETCHES TOP-MOST INDEX, PUTS IT INTO (IDX+HDR)
        PUSH IX
        PUSH DE
        LD IX,(TOP+PTR)
        DEC IX
        DEC IX
        LD E,(IX)
        LD D,(IX+1)
        LD (IDX+HDR),DE
        POP DE
        POP IX
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CNTSTK:
```

```
;COUNTS # OF NODES ON INHSTK, FROM CONS TO TOP
        CALL SAVREG
        LD HL,(TOP+PTR)
        LD DE,(BAS+PTR)
        OR 0
        SBC HL,DE
        LD DE,2
        SBC HL,DE
        SRL H
        RR  L
        LD (CNTIMM),HL
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
TOTSTK:
;ADDS UP # OF ALL BYTES STORED IN NODES
;FROM CONS TO TOP, STORES SUM IN (LENSUM)
        CALL SAVREG
;BC = # OF NODES ON CONS STK
        LD BC,(CNTIMM)
;IX = PTR TO NODES ON STK
        LD IX,(BAS+PTR)
        INC IX
        INC IX
;HL = SUM OF BYTES IN NODES
        LD HL,0
TS.1:   DEC C
        JP P,TS.2
        DEC B
        JP M,TS.3
TS.2:   LD E,(IX)
        LD D,(IX+1)
        INC IX
        INC IX
        LD (IDX+HDR),DE
;FETCH EACH NODE TO GET ITS LENGTH
        CALL FETCH
        LD DE,(SPC+HDR)
        ADD HL,DE
        JP TS.1
TS.3:   LD (LENSUM),HL
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKSTK:
;CHECKS ALL NODES ON CONS STK TO SEE IF THEY
;ARE SAME AS (TYPIMM)
        CALL SAVREG
;BC = # OF NODES ON CONS STK
```

```
        LD BC,(CNTIMM)
;IX = PTR TO NODES ON STK
        LD IX,(BAS+PTR)
        INC IX
        INC IX
CS.1:   DEC C
        JP P,CS.2
        DEC B
        JP M,CS.3
CS.2:   LD E,(IX)
        LD D,(IX+1)
        INC IX
        INC IX
        LD (IDX+HDR),DE
;FETCH EACH NODE TO GET ITS TYPE
        CALL FETCH
        LD A,(TYP+HDR)
        LD E,A
        LD A,(TYPIMM)
        SUB E
;IF ANY DON'T MATCH, RET WITH A .NE. 0
        JP NZ,CS.3
        JP CS.1
CS.3:   CALL RSTREG
        RET
;
LUP1:   LD BC,(STLUP1)
        DEC C
        JP P,LUP11
        DEC B
LUP11:  LD (STLUP1),BC
        RET
;
STLUP1: DW      0
;
CNTIMM: DW      0       ;NODES ON STK
LENSUM: DW      0       ;SUM OF THEIR LENGTHS
IDXIMM: DW      0       ;IDX OF IMM NODE
TYPIMM: DB      0       ;ITS TYP
        END
;........................................
```

```
;
;  6 OCT 83
; 21 DEC 83 - REMOVED DEAD WOOD MODULES AND STORAGE AREAS
;
        TITLE IONS A/O 21 DEC 83
;
GLOBAL PRNUM,PRSYM,PRBUL
GLOBAL MAKNUM,OUTNUM
GLOBAL MAKSYM,OUTSYM
GLOBAL OUTBUL
GLOBAL INPBUF
EXTERNAL GETATR,HDR,ALLOC,ATR
EXTERNAL PRCON
EXTERNAL ASCBCD,BCDASC
EXTERNAL SAVREG,RSTREG
;
                        .XLIST
        MACLIB EQATMO
        EQUATES
                        .LIST
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
MAKNUM:
;CONVERTS ASCII CHARACTERS IN INPBUF TO FXP NODE
        CALL SAVREG
        LD DE,(IDX+HDR)
        LD (ZZ+IDX),DE
MN.1:
;DEFAULT SIGN IS '+'
        LD A,POSFXP
        LD (SIGN),A
;SET FLAGS AND DIG COUNTS TO 0
        LD BC,0
        LD (DECPT),BC
        LD (DECCNT),BC
;GET CNT OF CHAR READ
        LD HL,CHCNT
        LD C,(HL)
;SET DE TO BUFFER FOR DIGITS
        LD DE,DIGBUF
;THIS THE LOOP WHICH EXAMINES ALL CHARS
IN.1:
;DEC CHAR COUNT UNTIL END OF BUFFER REACHED
        DEC C
        JP M,IN.4
;GET NEXT CHAR
        INC HL
        LD A,(HL)
;IF SPACE OR '+', SKIP
        CP ' '
        JP Z,IN.1
        CP '+'
        JP Z,IN.1
;IF '-', CHANGE SIGN
        CP '-'
        JP Z,IN.2
;IF '.', SET DECPT FLAG AND DECCNT
        CP '.'
        JP Z,IN.3
;CHECK IF BETWEEN 0-9
        CALL CHKRNG
        JP NZ,IN.11
;IF DIGIT, STORE IN DIGBUF
        INC B
        LD (DE),A
        INC DE
;GET NEXT CHAR
        JP IN.1
;
;SETS SIGN
IN.2:   LD A,NEGFXP
        LD (SIGN),A
        JP IN.1
;SETS DECPT FLAG AND DECCNT
IN.3:
;FIRST CHECK IF '.' FOUND ALREADY
        LD A,(DECPT)
        CP 0
        JP NZ,IN.11
;ELSE SET (DIGCNT)=B
        LD A,B
        LD (DIGCNT),A
        LD B,0
        LD A,1
        LD (DECPT),A
        JP IN.1
;LAST CHAR FOUND, SO COMPUTE # OF DIGITS
;TO RIGHT OF DEC PT
;FIRST SEE IF '.' READ
;IF NOT, SKIP PAST DECCNT COMPUTATION
IN.4:   LD A,(DECPT)
        CP 0
        JP Z,IN.7
;'.' WAS READ, SO GET # OF TRAILING DIGITS
;IF DECCNT ODD, STORE TRAILING '0'
IN.5:   BIT 0,B
        JP Z,IN.6
        INC B
        LD A,'0'
        LD (DE),A
;SAVE IN (DECCNT)
IN.6:   LD A,B
;DIVIDE DEC COUNT BY 2
        SRL A
        LD (DECCNT),A
        LD A,(DIGCNT)
        LD B,A
IN.7:
        LD DE,DIGBUF
        BIT 0,B
        JP Z,IN.8
        INC B
        DEC DE
IN.8:   LD A,B
;DIVIDE LEAD COUNT BY 2
        SRL A
        LD (DIGCNT),A
;SAFE TO USE SAME BUFFER FOR ASC->HEX CONVERSION
IN.9:   LD HL,INPBUF
        LD BC,(DIGCNT)
        ADD A,B
        LD C,A
        LD B,0
;CONVERT CHARS TO PACKED BCD
        CALL ASCBCD
;SET NODE TYPE
        LD A,(SIGN)
        LD (TYP+ZZ),A
;ADD 2 FOR DIG COUNTS
        INC C
        INC C
        LD (SPC+ZZ),BC
;ALLOCATE THE NODE
        CALL ALOCZ
;GET THE FIRST DATA ADDR
        LD HL,(FST+ZZ)
;STORE DIG COUNTS
        LD BC,(DIGCNT)
        LD (HL),C
        INC HL
        LD (HL),B
        INC HL
        LD A,C
        ADD A,B
        LD C,A
        LD B,0
;DE POINTS TO DESTINATION, HL TO DIGBUF
        EX DE,HL
        LD HL,INPBUF
;BLOCK MOVE TO STORE NUMBER
        LDIR
;
        CALL RSTREG
        RET
;
;ERROR MSG
IN.11:  LD DE,$NM.2
        CALL PRCON
        CALL RSTREG
        RET                     ; RETURN TO FORTH
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
PRNUM:
OUTNUM: ;FETCHES A FIXED POINT NUMBER FROM STORAGE
;AND PRINTS IT AT CONSOLE.
        CALL SAVREG
;GET THE INHER. NODE ADDR
        CALL GET10
        LD IX,ZZ
;CHECK IF FIXED PT NUMBER
        LD A,(IX+TYP)
        AND OFOH
        CP NUMBR
        JP NZ,ON.10
;IF OK, GET DEC AND DIG COUNTS
ON.1:
;STORE SIGN IN PRINT BUFFER
        LD DE,INPBUF
;SET SIGN
        LD A,(IX+TYP)
        LD B,'+'
        CP POSFXP
        JP Z,ON.6
        LD B,'-'
ON.6:   LD A,B
        LD (DE),A
;GET WHL,FRC COUNTS
        LD HL,(ZZ+FST)
        LD A,(HL)
        LD B,A
        LD (DIGCNT),A
        INC HL
        LD A,(HL)
        LD (DECCNT),A
        ADD A,B
        LD C,A
        LD B,0
;CONVERT TO ASCII CHARS.
        LD DE,DIGBUF
        INC HL
        CALL BCDASC
;SET HL TO CONVERTED DIGITS
        EX DE,HL
        LD DE,INPBUF+1
;FIGURE OUT DECIMAL PLACE
        LD A,(DIGCNT)
        LD C,A
        LD B,0
;IF NO LEADING DIGITS, SKIP AHEAD
        CP 0
        JP Z,ON.2
;ELSE, PRINT DIGITS
        SLA C
;SKIP LEADING ZERO
        LD A,(HL)
        CP '0'
        JP NZ,ON.3
        INC HL
        DEC C
ON.3:
        LDIR
ON.2:
;IF NO DIGITST OT RIGHT OF DEC PT, QUIT
        LD A,(DECCNT)
        CP 0
        JP Z,ON.5
;ELSE PRINT DEC PT AND CONTINUE
        LD C,A
        LD A,'.'
        LD (DE),A
        INC DE
        SLA C
        LDIR
;CLEAN OFF ANY TRAILING DIGITS
ON.9:   DEC DE
        LD A,(DE)
        CP '0'
        JP Z,ON.9
        INC DE
;FINISH BY PRINTING  DIGBUF
ON.5:   LD HL,CRLF$
        LDI
        LDI
        LDI
        LD DE,INPBUF
        CALL PRCON
ON.11:
        CALL RSTREG
        RET
;
ON.10:  LD DE,$NM.4
        CALL PRCON
        JP ON.11
;
CRLF$:  DB      ODH,OAH,'$'
$NM.2:
        DB 'ILLEGAL CHARACTER FOUND IN FIXED PT NUMBER $'
$NM.4:
        DB 'CANNOT PRINT NUMBER; NOT OF TYPE FIXED PT $'
$NM.9:
        DB 'NODE FOUND TO BE "NIL" $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;
MAKSYM:
;STORES CHARS IN INPBUF IN A NODE
        CALL SAVREG
        LD DE,(IDX+HDR)
        LD (ZZ+IDX),DE
MS.1:
;GET CNT OF CHAR READ
        LD B,0
        LD HL,INPBUF-1
        LD C,(HL)
;JMP AROUND CODE WHICH CHOPS LEADING/TRAILING BLANKS
        JP IS.99
;SET DE TO SAME BUFFER
        LD DE,INPBUF
;SKIP OVER LEADING BLANKS
IS.1:
;DEC CHAR COUNT UNTIL END OF BUFFER REACHED
        DEC C
        JP M,IS.2
;GET NEXT CHAR
        INC HL
        LD A,(HL)
;IF SPACE SKIP
        CP ' '
        JP Z,IS.1
;STORE THE REST
        LDIR
;NOW BACK UP OVER TRAILING BLANKS
IS.2:   DEC HL
        LD A,(HL)
        CP ' '
        JP Z,IS.2
;LAST CHAR FOUND, COMPUTE ACTUAL LENGTH
        LD DE,INPBUF-1
        AND 0
        SBC HL,DE
        LD C,L
        LD B,H
IS.99:
        LD (SPC+ZZ),BC
;DEFINE TYP
        LD A,SYMBL
        LD (TYP+ZZ),A
;(IDX+ZZ) MUST BE DEFINED, ALLOC THE NODE
        CALL ALOCZ
;STORE THE CHARACTERS
        LD DE,(FST+ZZ)
        LD HL,INPBUF
        LD BC,(SPC+ZZ)
        LDIR
;ALL DONE
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
PRSYM:
OUTSYM:
;PRINTS A SYMBOLIC ATOM AT CONSOLE
        CALL SAVREG
;GET THE INHER. NODE INDEX
        CALL GET10
        LD IX,ZZ
;CHECK IF SYMBL
        CALL CHKSYM
        JP NZ,OS.10
;IF OK, COPY NODE TO INPBUF
        LD DE,INPBUF
        LD HL,(FST+ZZ)
        LD BC,(SPC+ZZ)
        LDIR
;ADD $ TO END OF STRING, PRINT
        LD HL,$DLR
        LDI
        LD DE,INPBUF
        LD C,9
        CALL 5
        CALL RSTREG
        RET
OS.10:  LD DE,$SY.4
        CALL PRCON
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKSYM:
;CHECKS IF CURRENT NODE (IDX+ZZ) IS SYMBOLIC ATOM
;RETURNS "Z" IF SYMBL, "NZ" IF NOT
        LD A,(TYP+ZZ)
        AND OFOH
        CP SYMBL
        RET
;
$SY.4:  DB 'CANNOT PRINT: NOT OF TYPE SYMBL $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
PRBUL:
OUTBUL:
;PRINTS VALUE OF A BOOLN NODE AT CONSOLE
        CALL SAVREG
;GET THE INHER. NODE INDEX
        CALL GET10
        LD      IX,ZZ
```

```
;CHECK IF BOOLN
        CALL CHKBUL
        JP NZ,OB.5
;READ VALUE OF NODE
        LD IX,(FST+ZZ)
        LD A,(IX)
        CP TRUE
        JP Z,OB.1
        CP FALSE
        JP Z,OB.2
        JP OB.3
OB.1:   LD DE,$BL.5
        JP OB.4
OB.2:   LD DE,$BL.6
        JP OB.4
OB.3:   LD DE,$BL.7
        JP OB.5
OB.4:
        CALL PRCON
        CALL RSTREG
        RET
OB.5:   CALL PRCON
        CALL    RSTREG
        RET
;
$BL.5:  DB      ' BOOLEAN VALUE = TRUE '
        DB      ODH,OAH,'$'
$BL.6:  DB      ' BOOLEAN VALUE = FALSE '
        DB      ODH,OAH,'$'
$BL.7:  DB 'TYP NOT BOOLN, OR VALUE NOT T/F $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKBUL:
        LD A,(TYP+ZZ)
        AND OFOH
        CP BOOLN
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GET10:
;GETS 1 INHERITED NODE FROM INHSTK
;AND DEFINES ZZ HDR BLCK
        CALL SAVREG
        LD BC,1
        LD (NINH+ATR),BC
        LD BC,0
        LD (NSYN+ATR),BC
        LD BC,ZATR
        LD (DESC+ATR),BC
        CALL GETATR
;COPY HDR INFO TO ZZ
```

```
        LD HL,(ZATR)
        LD DE,ZZ
        LD BC,11
        LDIR
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GET01:
;GETS 1 SYN. ATTR. INDEX OFF INHSTK
;DEFINES ONLY IDX IN ZZ
        CALL SAVREG
        LD BC,0
        LD (NINH+ATR),BC
        LD BC,1
        LD (NSYN+ATR),BC
        LD BC,ZATR
        LD (DESC+ATR),BC
        CALL GETATR
;COPY IDX
        LD HL,(ZATR)
        LD DE,ZZ
        LDI
        LDI
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ALOCZ:
;ALLOCATES ONE NODE (Z) USING INFO
;IN ZZ.  THEN SAVES ALL HDR INFO INZZ
        CALL SAVREG
        LD DE,IDX+HDR
        LD HL,ZZ
        LD BC,5
        LDIR
        CALL ALLOC
;NOW SWITCH DE,HL AND COPY BACK FROM ATRBLK
        LD BC,6
        EX DE,HL
        LDIR
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;DATA BUFFER AREAS
;
ZATR:   DW      0
ZZ:     DS      12
CHCNT:  DB      0
INPBUF: DS      0FFH
```

```
$CRLF:   DB      ODH,OAH,'$'
$DLR:    DB      '$'
;
SIGN:    DB      POSFXP
DECPT:   DB      0
DIGCNT:  DB      0
DECCNT:  DB      0
LDZFLG:  DB      0
LD0:     DB      30H
DIGBUF:  DS      OFEH
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CHKRNG: ;CHECKS RANGE OF CHAR TO SEE IF A DIGIT
        CP 3AH
        RET P
        CP 30H
        RET M
        PUSH BC
        LD B,A
        CP B
        POP BC
        RET
;
        END             ; END OF IONS
;.....................................................
```

```
; 6 OCT 83
; 21 DEC 83 - REMOVED DEADWOOD MODULES AND STORAGE AREAS
; 02 FEB 84 - FIXED NORMAL
; 06 FEB 84 - FIXED ALOCZ TO STORE +0 FOR ZERO NUMERIC NOD
;
        TITLE MATH A/O 06 FEB 84
;
GLOBAL AD,SB,ML,DV
GLOBAL AB,NG,INT
EXTERNAL GETATR,ALOSYN,ATR,HDR
EXTERNAL CMPNUM
EXTERNAL ALLOC,PRLINE,SAVREG,RSTREG
;
        .XLIST
        MACLIB EQATMO
        EQUATES
                                .LIST
        WHL EQU LST+2
        FRC EQU WHL+1
        TOT EQU FRC+1
        MSB EQU TOT+1
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
AD:     LD A,00H
        JP OP.0
SB:     LD A,10H
        JP OP.0
;
;ESTABLISH OPCODE FROM OPERAND PROPERTIES
OP.0:   LD (OPCODE),A
;PRESET RESULT TYP TO POSFXP, AND OPFLG TO '+'
        LD A,POSFXP
        LD (TYP+ZZ),A
        LD A,'+'
        LD (OPFLG),A
;UNSTACK 3 NODE INDICES FROM INHSTK (X,Y,Z)
        CALL GETXYZ
;LOOK AT SIGNS OF X,Y TO FURTHER DEFINE OPCODE
        LD A,(XX+TYP)
        AND 02H
;IF X>0, ADD 4, ELSE ADD NOTHING
        JP Z,OP.01
        LD A,(OPCODE)
        ADD A,4
        LD (OPCODE),A
OP.01:  LD A,(YY+TYP)
        AND 02H
;IF Y>0, ADD 2, ELSE ADD NOTHING
        LD A,(OPCODE)
        JP Z,OP.02
        ADD A,2
        LD (OPCODE),A
OP.02:
;
;THIS IS THE ADD/SUB SECTION
;
;CALL COMPAR: IF !X!<!Y!, ADD 1 TO OPCODE,
;AND SWITCH IX,IY
;
;SET IX/Y TO ADDRS OF X/Y
        LD IX,(XX+ADR)
        LD IY,(YY+ADR)
;TEMPORARILY SET BOTH TYP'S TO POSFXP
        LD A,POSFXP
        LD (IX+TYP),A
        LD (IY+TYP),A
        CALL CMPNUM
;RESTORE NODE TYPES
        LD A,(XX+TYP)
        LD (IX+TYP),A
        LD A,(YY+TYP)
        LD (IY+TYP),A
        LD A,(OPCODE)
        LD IX,XX
        LD IY,YY
        JP P,OP.10
        ADD A,1
        LD IX,YY
        LD IY,XX
OP.10:  LD (OPCODE),A
;IF OPCODE =0000011X OR 0001010X, RESULT IS POS, ADD X,Y.
        CP 00000111B
        JP Z,OP.11
        CP 00000110B
        JP Z,OP.11
        CP 00010101B
        JP Z,OP.11
        CP 00010100B
        JP Z,OP.11
;IF OPCODE=0000000X OR 0001001X, RESULT IS NEG, ADD X,Y.
        CP 00000000B
        JP Z,OP.12
        CP 00000001B
        JP Z,OP.12
        CP 00010010B
        JP Z,OP.12
        CP 00010011B
        JP Z,OP.12
;IF OPCODE=00000100 OR 00010110, RESULT POS., SUBTRACT X-Y
        CP 00000100B
        JP Z,OP.13
        CP 00000011B
        JP Z,OP.13
        CP 00010110B
        JP Z,OP.13
        CP 00010001B
        JP Z,OP.13
;FOR ALL OTHER OPS, RESULT NEG, SUB X,Y
;ADJUST RESULT TYP, OPFLG BEFORE ADD/SUB
OP.14:  LD A,NEGFXP
        LD (TYP+ZZ),A
OP.13:  LD A,'-'
        LD (OPFLG),A
        JP OP.20
OP.12:  LD A,NEGFXP
        LD (TYP+ZZ),A
OP.11:  JP OP.20
;
OP.20:
;MAKE ZFRC LARGER OF XFRC,YFRC; PUSH (XFRC-YFRC)
        LD B,(IX+FRC)
        LD C,(IY+FRC)
        LD A,B
        CP C
        LD A,C
        JP M,OP.21
        LD A,B
OP.21:  LD (ZZ+FRC),A
        LD L,A
;(SPC+HDR) = L + LARGER OF XWHL,YWHL +3
        LD B,(IX+WHL)
        LD C,(IY+WHL)
        LD A,B
        CP C
        LD A,C
        JP M,OP.22
        LD A,B
OP.22:  INC A
        LD (ZZ+WHL),A
        ADD A,L
        LD (TOT+ZZ),A
;ZERO OUT THE Z NODE
        LD HL,RESULT
        LD (MSB+ZZ),HL
        CALL ZEROZ
;COPY X->Z
        CALL COPYXZ
;ALIGN Y AND Z FOR ADD/SUB
        LD HL,(LST+ZZ)
        LD A,(FRC+ZZ)
        SUB (FRC+IY)
        LD C,A
        LD B,0
        OR 0
        SBC HL,BC
        EX DE,HL
        LD L,(IY+LST)
        LD H,(IY+LST+1)
        LD C,(IY+TOT)
;READY TO CALL BCDADD/SUB, DEPENDING ON (OPFLG)
        LD A,(OPFLG)
        CP '+'
        JP NZ,OP.30
        CALL BCDADD
        JP OP.31
OP.30:  CALL BCDSUB
OP.31:
;REMOVE ANY LEADING,TRAILING ZEROS
        CALL NORMAL
;STORE RESULT
        CALL ALOCZ
        CALL STRSLT
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ML:
; Z = X * Y (DECIMAL FRACTIONS ALLOWED)
        CALL GETXYZ
        LD IX,XX
        LD IY,YY
        LD A,(IX+WHL)
        ADD A,(IY+WHL)
        INC A
        LD (ZZ+WHL),A
        LD B,A
        LD A,(IX+FRC)
        ADD A,(IY+FRC)
        LD (ZZ+FRC),A
        ADD A,B
        LD (ZZ+TOT),A
        LD HL,RESULT
        LD (MSB+ZZ),HL
        CALL ZEROZ
;SET DE TO PT. TO LST OF Z, HL TO LSB OF X
;IY TO LST OF Y
;C=# OF BYTES IN X, B=# OF BYTES IN Y
        LD DE,(ZZ+LST)
        LD HL,(XX+LST)
        LD C,(IX+TOT)
        LD B,(IY+TOT)
        LD IY,(YY+LST)
;THIS IS THE BIG MULTIPLY LOOP, REPEATED NY TIMES
MUL.2:  DEC B
        JP M,MUL.4
        LD (TEMP),BC
;LET B=# OF TIMES TO ADD X TO Z
        LD A,(IY)
        CALL BCDHEX
        LD B,A
```

```
MUL.3:  DEC B
        JP M,MUL.31
        CALL BCDADD
        JP MUL.3
MUL.31: LD BC,(TEMP)
        DEC IY
        DEC DE
        JP MUL.2
;ALL DONE
MUL.4:
;REMOVE ANY LEADING,TRAILING ZEROS
        CALL NORMAL
;STORE RESULT
        CALL SIGNMD
        CALL ALOCZ
        CALL STRSLT
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
DV:
; Z = X / Y (DECIMAL FRACTIONS ALLOWED)
        CALL GETXYZ
        LD IX,XX
        LD IY,YY
;ESTIMATE # OF LEAD ZEROES IN Z=WY-WX
        LD A,(IY+WHL)
        SUB (IX+WHL)
        DEC A
;COUNT LEAD ZEROES IN Y: DEC LZ
        LD BC,(TOT+YY)
        LD B,A
        LD HL,(MSB+YY)
DV.00:  DEC C
        JP M,DV.01
        LD A,(HL)
        CP 0
        JP NZ,DV.01
        DEC B
        INC HL
        JP DV.00
DV.01:  INC C
        LD A,C
        LD (TOT+YY),A
        LD (MSB+YY),HL
;NOW LOOK FOR LEAD ZEROES IN X: INC LZ
        LD HL,(MSB+XX)
        LD A,(TOT+XX)
        LD C,A
DV.02:  DEC C
        JP M,DV.03
        LD A,(HL)
        CP 0
        JP NZ,DV.03
        INC B
        INC HL
        JP DV.02
DV.03:  INC C
        LD A,C
        LD (TOT+XX),A
        LD (MSB+XX),HL
;WZ=E=-LZ, FZ=D=LZ+NSIG-WZ
        LD A,0
        SUB B
        LD E,A
;
;SET E,B TO 0 IF NEG.
        LD A,0
        CP B
        JP M,DV.21
        LD B,0
DV.21:
        CP E
        JP M,DV.23
        LD E,0
DV.23:  LD A,B
        LD (LZ),A
        LD A,(NSIG)
        ADD A,B
        SUB E
        JP P,DV.24
        LD A,0
DV.24:  LD (FRC+ZZ),A
        LD A,E
        LD (WHL+ZZ),A
;LEAVE LZ+3 LEAD ZEROES IN RESULT
        LD HL,RESULT
        LD BC,(LZ)
        LD B,0
        LD IX,RESULT
        ADD IX,BC
        INC BC
        INC BC
        LD DE,RESULT+1
        LD (HL),0
        LDIR
        LD (MSB+ZZ),DE
;COPY X => RESULT
        LD HL,(MSB+XX)
        LD BC,(TOT+XX)
        LD B,0
        DEC C
        JP M,DV.41
        INC C
        LDIR
DV.41:
```

```
;LEAVE NSIG-XTOT MORE ZEROES IN RESULT
        EX DE,HL
        LD A,(NSIG)
        LD BC,(TOT+XX)
        SUB C
        JP M,DV.43
        JP Z,DV.43
DV.42:  DEC A
        JP M,DV.43
        LD (HL),0
        INC HL
        JP DV.42
DV.43:
;IX PTS TO RESULT BYTES IN Z
;MIN(YTOT,NSIG) = # OF BYTES TO SUB Y FROM Z
        LD A,(NSIG)
        LD BC,(TOT+XX)
        CP C
        JP P,DV.44
        LD A,C
DV.44:  INC A
        LD (TSIG),A
DV.50:  LD A,(TSIG)
        DEC A
        JP Z,DV.59
        LD (TSIG),A
        CP (IY+TOT)
        JP M,DV.51
        LD A,(YY+TOT)
DV.51:  LD C,A
;SET HL/DE TO LST OF Y/RESULT
        LD B,0
;UPDATE MSBZ
        LD HL,(ZZ+MSB)
        INC HL
        LD (ZZ+MSB),HL
        OR 0
        ADC HL,BC
        DEC HL
        DEC HL
        EX DE,HL
        LD HL,(MSB+YY)
        OR 0
        ADC HL,BC
        DEC HL
;NOW READY TO ENTER SUB LOOP
        LD (IX),0
        LD (IX+1),0
DV.54:
        CALL BCDSUB
        JP C,DV.55
        INC (IX)
        JP DV.54
```

```
;ADD Y BACK TO Z ONCE
DV.55:
        CALL BCDADD
        LD A,(IX)
        CALL HEX100
        LD (IX),A
        INC IX
        JP DV.50
DV.59:
;REMOVE ANY LEADING,TRAILING ZEROS
        CALL NORMAL
;STORE RESULT
        CALL SIGNMD
        CALL ALOCZ
        CALL STRSLT
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
AB:
;ABSOLUTE VALUE
        CALL GETXZ
        LD A,POSFXP
        LD (TYP+ZZ),A
        CALL IDXZ
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NG:
;NEGATION
        CALL GETXZ
        LD A,(TYP+XX)
        CP POSFXP
        LD A,NEGFXP
        JP Z,NG.1
        LD A,POSFXP
NG.1:   LD (TYP+ZZ),A
        CALL IDXZ
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
INT:
;RETURNS INTEGER PART
        CALL GETXZ
        LD A,0
        LD (FRC+XX),A
        LD A,(TYP+XX)
        LD (TYP+ZZ),A
        CALL IDXZ
        RET
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
HEX100:
;CONVERTS HEX # IN A TO DECIMAL
        LD C,A
        LD A,0
HEX.1:  DEC C
        JP M,HEX.2
        ADD A,1
        DAA
        JP HEX.1
HEX.2:  RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SIGNMD:
;SETS SIGN OF RESULT OF MUL/DIV
        LD A,(TYP+XX)
        LD BC,(TYP+YY)
        CP C
        LD A,POSFXP
        JP Z,SMD.1
        LD A,NEGFXP
SMD.1:
        LD (TYP+ZZ),A
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GETXZ:
;GETS X,Z: 1 INHER., 1 SYNTH.
        CALL SAVREG
        LD BC,1
        LD (NINH+ATR),BC
        LD (NSYN+ATR),BC
        LD BC,YATR
        LD IX,YATR
        JP GT.0
;
;
GETXYZ:
;DEFINES 16 BYTES OF DESCRIPTOR BLK OF
;X,Y AND DEFINES INDEX OF Z
        CALL SAVREG
        LD BC,2
        LD (NINH+ATR),BC
        LD BC,1
        LD (NSYN+ATR),BC
        LD BC,XATR
        LD IX,XATR
GT.0:
        LD (DESC+ATR),BC
        CALL GETATR
```

```
;COPY HDR BLKS OF X,Y TO XX,YY
;AND DEFINE OTHER FIXED PT PARAMETERS
        LD BC,(NINH+ATR)
        CALL STLUP1
        LD IY,XX
        LD DE,XX
GT.1:   CALL LUP1
        JP M,GT.2
        LD L,(IX)
        LD H,(IX+1)
;COPY 11 BYTES FROM ATRBLK TO LOCAL
        LD BC,11
        LDIR
;DEFINE OTHER PARAMS
        LD L,(IY+FST)
        LD H,(IY+FST+1)
        LD A,(HL)
        LDI
        ADD A,(HL)
        LDI
        LD (DE),A
;HL PTS TO MSB OF NUMBR
        LD (IY+MSB),L
        LD (IY+MSB+1),H
;STORE LST BYTE OF #
        LD C,A
        LD B,0
        OR 0
        ADC HL,BC
        DEC HL
        LD (IY+LST),L
        LD (IY+LST+1),H
;REPEATED ONLY ONCE MORE
        LD IX,YATR
        LD IY,YY
        LD DE,YY
        JP GT.1
GT.2:
        LD HL,(ZATR)
        LD DE,ZZ
        LDI
        LDI
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
IDXZ:
;MAKES IDENTICAL COPY, EXCEPT FOR TYPE
        LD BC,(WHL+XX)
        LD (WHL+ZZ),BC
        CALL ALOCZ
        LD BC,(TOT+ZZ)
        LD B,0
        LD A,0
        CP C
        JP P,IDXZ.1
        LD HL,(MSB+XX)
        LD DE,(MSB+ZZ)
        LDIR
IDXZ.1: RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ALOCZ:
;ALLOCATES RESULT (ZZ), AND STORES WHL,FRC
        CALL SAVREG
        LD BC,(WHL+ZZ)
        LD A,B
        ADD A,C
        CP      0               ; 2/6 CHECK FOR  DIGITS
        JP      NZ,AL.1         ; 2/6
        INC     A               ; 2/6 IF NO DIGITS, ADD ON
        LD      (WHL+ZZ),A      ; 2/6 STORE 1 IN WHL
AL.1:   LD (TOT+ZZ),A
        CP      1               ; 2/6 ONLY ONE DIGIT?
        JP      NZ,AL.2         ; 2/6
        LD      A,(RESULT)      ; 2/6 GET 1ST BYTE OF RESU
        CP      0               ; 2/6 IS DATA BYTE ZERO?
        JP      NZ,AL.2         ; 2/6
        LD      A,POSFXP        ; 2/6 IF YES ENSURE +0
        LD      (TYP+ZZ),A      ; 2/6
AL.2:   LD      A,(TOT+ZZ)      ; 2/6
        INC A
        INC A
        LD C,A
        LD B,0
        LD (SPC+ZZ),BC
        LD DE,IDX+HDR
        LD HL,ZZ
        LD BC,5
        LDIR
        CALL ALLOC
        EX DE,HL
        LD BC,6
        LDIR
;STORE WHL,FRC IN Z NODE
        LD DE,(ZZ+FST)
        LD HL,ZZ+WHL
        LDI
        LDI
;STORE MSB (DE) ADDR LOCALLY
        LD (MSB+ZZ),DE
        CALL RSTREG
        RET
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
STRSLT:
;COPIES RESULT TO Z NODE
        CALL SAVREG
        LD DE,(MSB+ZZ)
        LD HL,RESULT
        LD BC,(TOT+ZZ)
        LD B,0
        DEC C
        JP M,ST.1
        INC C
        LDIR
ST.1:   CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ZEROZ:
;ZEROS (TOT+ZZ) BYTES OF Z NODE
        CALL SAVREG
        LD HL,(MSB+ZZ)
        LD DE,(MSB+ZZ)
        INC DE
        LD (HL),0
        LD BC,(TOT+ZZ)
        DEC C
        JP M,ZRO.1
        JP Z,ZRO.1
        LD B,0
        LDIR
ZRO.1:
        LD (LST+ZZ),HL
        CALL RSTREG
        RET
;
;
COPYXZ:
;COPIES X -> Z, ALIGNING DECIMAL POINTS
        CALL SAVREG
        LD DE,(MSB+ZZ)
        INC DE
        LD C,(TOT+IX)
        LD B,0
        LD L,(MSB+IX)
        LD H,(MSB+IX+1)
        LDIR
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
BCDADD:
```

```
;ADDS 2 BCD NUMBERS, ONE IN Z NODE, OTHER IN Y
;DE/HL PT. TO LAST BYTES OF Z,Y
;C=# OF BYTES TO ADD
        CALL SAVREG
        OR 0
BCAD.0: DEC C
        JP M,BCAD.1
        LD A,(DE)
        ADC A,(HL)
        DAA
        LD (DE),A
        DEC HL
        DEC DE
        JP BCAD.0
BCAD.1: LD A,(DE)
        ADC A,0
        DAA
        LD (DE),A
        DEC DE
        JP C,BCAD.1
BCAD.2: CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
BCDSUB:
;SAME AS BCDADD, EYCEPT SUBTRACT
        CALL SAVREG
        OR 0
BCSB.0: DEC C
        JP M,BCSB.1
        LD A,(DE)
        SBC A,(HL)
        DAA
        LD (DE),A
        DEC HL
        DEC DE
        JP BCSB.0
BCSB.1: LD A,(DE)
        SBC A,0
        DAA
        LD (DE),A
BCSB.2: CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
BCDHEX:
;CONVERTS 2 BCD DIGITS IN A TO HEX
        CALL SAVREG
        LD HL,DUM
        LD (HL),A
        XOR A
        RLD
        LD C,A
        XOR A
BH.1:   DEC C
        JP M,BH.2
        ADD A,10
        JP BH.1
BH.2:   LD C,A
        XOR A
        RLD
        ADD A,C
        CALL RSTREG
        RET
DUM:    DB      0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NORMAL:
;REMOVES LEADING/TRAILING ZEROES FROM RESULT (Z).
        CALL SAVREG
        LD HL,RESULT
        LD BC,(WHL+ZZ)
;TAKE OFF ANY LEADING ZERO'S
NM.1:   DEC C
        JP M,NM.2
        LD A,(HL)
        CP 0
        JP NZ,NM.2
        INC HL
        JP NM.1
NM.2:   INC     C
        LD      A,C             ; 2/2
        LD (WHL+ZZ),A
        ADD A,B
        LD C,A
        DEC C
        JP      P,NM.3          ; 2/2
        JP      Z,NM.3          ; 2/2
        LD      (HL),0          ; 2/2
        INC     C               ; 2/2
        LD      A,1             ; 2/2
        LD      (WHL+ZZ),A      ; 2/2
NM.3:   INC     C               ; 2/2
        LD DE,RESULT
        LD B,0
        LDIR
;CHECK FOR ANY TRAILING ZERO'S
        EX DE,HL
        DEC HL
        LD BC,(WHL+ZZ)
NM.5:   DEC B
        JP M,NM.6
        LD A,(HL)
```

```
         CP 0
         JP NZ,NM.6
         DEC HL
         JP NM.5
NM.6:    INC B
         LD A,B
         LD (FRC+ZZ),A
NM.7:    CALL RSTREG
         RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
STLUP1:
         LD (CLUP1),BC
         RET
;
LUP1:
         LD (TEMP),BC
         LD BC,(CLUP1)
         DEC C
         JP P,LUP11
         DEC B
LUP11:   LD (CLUP1),BC
         LD BC,(TEMP)
         RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
                 .LIST
XATR:    DW      0
YATR:    DW      0
ZATR:    DW      0
XX:      DS      16
YY:      DS      16
ZZ:      DS      16
OPCODE:  DB      0
OPFLG:   DB      0
CLUP1:   DW      0
TEMP:    DW      0
AFTEMP:  DW      0
HLTEMP:  DW      0
LZ:      DW      0
NSIG:    DW      4
TSIG:    DW      0
RESULT:  DS      256
END
;END OF MATH.......................................
```

```
;
; 6 OCT 83
; 29 NOV 83 - MADE ALL RST 38H RETURN TO FORTH
; 21 DEC 83 - REMOVED ALL DEADWOOD MODULES AND STORAGE ARE
;
        TITLE RADX A/O 21 DEC 83
;
        GLOBAL BCDASC,ASCBCD
        GLOBAL HEXASC,HEXWRD,ASCHEX
        GLOBAL HEXBCD,BCDHEX,HEX100
        EXTERNAL SAVREG,RSTREG
        EXTERNAL PRLINE
;
                        .XLIST
        MACLIB EQATMO
                        .LIST
;
;BCDASC AND ASCBCD CONVERT BETWEEN STRINGS
;OF ASCII CHARS. 0-9 AND BCD NUMBERS
; UPON ENTRY AND EXIT..
; BC=# OF DIGITS TO BE CONVERTED
; HL=PTR TO STRING OF PACKED BCD DIGITS
; DE=PTR TO STRING OF ASCII CHARACTERS
;
ASCBCD:
        CALL SAVREG
ASBC.1: DEC C
        JP M,ASBC.2
        LD A,(DE)
        INC DE
        SUB 30H
        CALL RNGCHK
        RLD
        LD A,(DE)
        INC DE
        SUB 30H
        CALL RNGCHK
        RLD
        INC HL
        JP ASBC.1
ASBC.2: CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
BCDASC:
        CALL SAVREG
        LD (TEMP),DE
        LD IX,(TEMP)
BCAS.1: DEC C
        JP M,BCAS.2
        XOR A
        LD D,(HL)
```

```
         RRD
         CALL RNGCHK
         ADD A,30H
         LD (IX+1),A
         XOR A
         RRD
         CALL RNGCHK
         ADD A,30H
         LD (IX),A
         LD (HL),D
         INC IX
         INC IX
         INC HL
         JP BCAS.1
BCAS.2:  CALL RSTREG
         RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
RNGCHK:
;CHECKS THAT DIGIT IN RNG 0-9
         CP 10
         JP P,RC.1
         CP 0
         JP M,RC.1
         RET
RC.1:    LD DE,$RC.1
         CALL PRLINE
         RET               ; 11/29
;
$RC.1:   DB 'CHAR OR NUM OUT OF RANGE (0-9) $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
HEXASC:
;CONVERTS AN N-BYTE STRING OF HEX VALUES
;TO A 2N-BYTE STRING OF ASCII CHARACTERS
;BC=N, DE PTS TO BUFFER FOR ASCII CHARS,
;HL PTS TO FIRST HEX BYTE
         CALL SAVREG
         LD (TEMP),DE
         LD IY,(TEMP)
HXAS.0:  DEC C
         JP M,HXAS.3
         LD (IY),' '
         INC IY
         CALL HXCONV
HXAS.1:  LD (IY+1),A
         CALL HXCONV
HXAS.2:  LD (IY),A
         INC IY
         INC IY
         LD (HL),E
```

```
        INC HL
        JP HXAS.0
HXAS.3: CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
HEXWRD:
;CONVERTS AN N-BYTE STRING OF HEX VALUES TO
;A 2N-BYTE STRING OF ASCII CHARACTERS
;BYTES DISPLAYED AS HEX WORDS
;BC=N, DE PTS TO BUFFER FOR ASCII CHARS,
;HL PTS TO FIRST HEX BYTE
        CALL SAVREG
        LD (TEMP),DE
        LD IY,(TEMP)
HXWD.0:
        DEC C
        JP M,HXWD.3
        LD (IY),' '
        INC IY
        INC HL
        LD E,(HL)
        CALL HXCONV
HXWD.1: LD (IY+1),A
        CALL HXCONV
HXWD.2: LD (IY),A
        INC IY
        INC IY
        LD (HL),E
        DEC HL
        LD E,(HL)
        CALL HXCONV
        LD (IY+1),A
        CALL HXCONV
        LD (IY),A
        INC IY
        INC IY
        LD (HL),E
        INC HL
        INC HL
        JP HXWD.0
HXWD.3:
        LD (IY),'$'
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
HXCONV:
;DOES ACTUAL CONVERSION OF NIBBLES IN (HL)
        XOR A
        RRD
        ADD A,30H
        CP 3AH
        RET M
        ADD A,7
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ASCHEX:
;CONVERTS A 2N-BYTE STRING OF ASCII CHARACTERS
;TO AN N-BYTE STRING OF HEX BYTES
;BC=2N, DE PTS TO BUFFER FOR HEX BYTES,
;HL POINTS TO ASCII CHARACTERS
        CALL SAVREG
        LD (TEMP),HL
        LD IY,(TEMP)
        EX DE,HL
        SRL C
ASHX.0:
        DEC C
        JP M,ASHX.3
        LD A,(IY+1)
        SUB 30H
        CP 11H
        JP M,ASHX.1
        SUB 7
ASHX.1: RRD
        LD A,(IY)
        SUB 30H
        CP 11H
        JP M,ASHX.2
        SUB 7
ASHX.2: RRD
        INC IY
        INC IY
        INC HL
        JP ASHX.0
ASHX.3:
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;HEXBCD: CONVERTS 2-BYTE HEX NUMBER TO 3-BYTE BCD NUMBER
;BCDHEX: DOES REVERSE
;HEX100 CONVERTS HEX BYTE (IN A) TO DEC. BYTE
;
HEXBCD:
;UPON ENTRY HL=HEX NUMBER,
;DE PTS TO BUFFER FOR 3 BCD DIGITS
        CALL SAVREG
        ;P10TAB IS A TABLE OF POWERS OF TEN
        LD IY,P10TAB
        LD (TEMP),DE
        EX DE,HL
        LD (HL),0
        LD C,5
HB.0:   EX DE,HL
        XOR A
        LD E,(IY)
        LD D,(IY+1)
HB.1:   OR A
;SUBTRACT POWER OF TEN
        SBC HL,DE
;KEEP DIVIDING UNTIL NC
        JP C,HB.2
        INC A
        JP HB.1
;RESTORE HL TO POS.
HB.2:   ADD HL,DE
;SAVE BCD DIGIT
        EX DE,HL
        LD HL,(TEMP)
        RLD
        BIT 0,C
        JP Z,HB.3
        INC HL
        LD (HL),0
HB.3:   LD (TEMP),HL
        INC IY
        INC IY
        DEC C
        JP NZ,HB.0
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
BCDHEX:
;CONVERTS UP TO 3 BCD BYTES TO A 16-BIT HEX #
;ON RETURN, DE PTS TO HEX #
;FIRST BYTE, C=# OF BYTES, 0<C<=3
;CHECK C
        CALL SAVREG
        LD A,C
        CP 0
        JP Z,BCHX.8
        CP 4
        JP P,BCHX.8
;LET HL PT TO LAST BYTE
        LD B,0
        ADD HL,BC
;HL'=ACCUMULATOR, ZERO IT
        EXX
        LD HL,0
        EXX
        AND 0
;LET IY PT TO POWER OF 10 TO ADD
        LD IY,P10TAB+10
;SET UP BIG LOOP TO MULTIPLY BCD BYTES
BH.1:   DEC C
        JP M,BH.6
        DEC IY
        DEC IY
        DEC HL
;SAVE BYTE (HL) FOR LATER
        LD A,(HL)
        LD (TEMP),A
        LD A,0
        RRD
;GO TO HL',DE'
        EXX
;MAKE DE' THE ADDEND
        LD E,(IY)
        LD D,(IY+1)
BH.2:   DEC A
        JP M,BH.3
        ADC HL,DE
        JP BH.2
BH.3:   DEC IY
        DEC IY
        LD E,(IY)
        LD D,(IY+1)
;PUT UPPER 4 BITS IN A
        LD A,0
        EXX
        RLD
        EXX
BH.4:   DEC A
        JP M,BH.5
        ADC HL,DE
        JP BH.4
;DONE WITH THIS (HL), RESTORE IT
BH.5:   EXX
        LD A,(TEMP)
        LD (HL),A
        JP BH.1
BH.6:   EXX
        LD (HEXNUM),HL
        EXX
        LD HL,HEXNUM
        LDI
        LDI
        JP C,BCHX.9
BH.7:   CALL RSTREG
        RET
HEXNUM: DW      0
;
BCHX.8: LD DE,$BCHX8
```

```
        JP QUIT
BCHX.9: LD DE,$BCHX9
        CALL PRLINE
        JP BH.7
$BCHX8: DB 'CANT CONVERT >3 OR <0 BCD BYTES TO HEX $'
$BCHX9: DB 'HEX # >64K IN BCDHEX $'
;
QUIT:   CALL PRLINE
        JP      BH.7             ; RETURN TO FORTH
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
HEX100:
;CONVERTS HEX BYTE IN A (<100)
;TO BCD BYTE IN A
        CP 100
        JP P,HX.1
        CP 0
        JP M,HX.1
;USE BCBUF AND CALL HEXBCD
        CALL SAVREG
        LD H,0
        LD L,A
        LD DE,BCBUF
        CALL HEXBCD
;RETURN BCD BYTE IN A
        LD DE,BCBUF+2
        LD A,(DE)
        CALL RSTREG
        RET
;
HX.1:   LD DE,$HX100
        CALL PRLINE
        RET
;
$HX100: DB 'HEX VALUE OUT OF (0-99) $'
;
TEMP:   DW      0
P10TAB: DW      10000
        DW       1000
        DW        100
        DW         10
        DW          1
;
HXNUM:  DW      1111
BCBUF:  DS      3
;
        END
;.................................................
```

```
;
;6 OCT 83
; 29 NOV 83 - MADE ALL RST 38H RETURN TO FORTH
; 21 DEC 83 - REMOVED DEADWOOD MODULES AND STORAGE AREAS
;
        TITLE RELN A/O 21 DEC 83
;
GLOBAL CMPNUM,CMPSYM,CMPSEQ
GLOBAL EQ,NE,LT,LE,GT,GE
EXTERNAL GETATR,ALOSYN,ATR,HDR
EXTERNAL SAVREG,RSTREG,PRLINE,FETCH
;
                        .XLIST
        MACLIB EQATMO
                EQUATES
                        .LIST
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
EQ:             ;COMPARES 2 NODES FOR EQUALITY
;1) COMPARE THEIR TYPES
;2) IF NON-NIL ATOMS OR SEQUENCES, COMPARE THEIR LENGTHS
;3) IF ATOMS, COMPARE EACH DIGIT OR CHAR
;4) IF SEQUENCE, COMPARE EACH ELEMENT
;RETURNS BOOLEAN
;GET X,Y,Z NODES, DEFINE XX,YY BLKS
        CALL XYZBUL
;COMPARE X,Y
        CALL COMPAR
        JP NZ,RELNO
;STATEMENTS BELOW ARE COMMON TO ALL RELN PRIMITIVES
RELYES: LD A,TRUE
        JP RELQU
RELNO:  LD A,FALSE
RELQU:  LD (RESTYP),A
;STORE THE BOOLEAN NODE (Z)
        CALL STORZ
        RET
;
RESTYP: DB      0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NE:             ;COMPARES 2 NODES FOR INEQUALITY.
;CALLS EQ, ABOVE, AND COMPLEMENTS RESULT.
;GET X,Y,Z NODES, DEFINE XX,YY BLKS
        CALL XYZBUL
;COMPARE X,Y
        CALL COMPAR
        JP NZ,RELYES
        JP RELNO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
LT:     ;COMPARES 2 NODES FOR 'LESS-THAN' ORDERING REL.
;GET X,Y,Z NODES, DEFINE XX,YY BLKS
        CALL XYZBUL
;COMPARE X,Y
        CALL COMPAR
        JP M,RELYES
        JP RELNO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
LE:     ;COMPARES 2 NODES FOR 'LESS-THAN OR EQUAL-TO'
;GET X,Y,Z NODES, DEFINE XX,YY BLKS
        CALL XYZBUL
;COMPARE X,Y
        CALL COMPAR
        JP M,RELYES
        JP Z,RELYES
        JP RELNO
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GT:     ;COMPARES 2 NODES FOR 'GREATER-THAN' ORDERING.
;GET X,Y,Z NODES, DEFINE XX,YY BLKS
        CALL XYZBUL
;COMPARE X,Y
        CALL COMPAR
        JP Z,RELNO
        JP M,RELNO
        JP RELYES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GE:     ;COMPARES 2 NODES FOR 'GREATER-THAN OR
;EQUAL-TO' ORDERING.
;GET X,Y,Z NODES, DEFINE XX,YY BLKS
        CALL XYZBUL
;COMPARE X,Y
        CALL COMPAR
        JP M,RELNO
        JP RELYES
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
XYZBUL:
;GETS OPERAND AND RESULT INDICES OFF INHSTK
        LD BC,2
        LD (NINH+ATR),BC
        LD BC,1
        LD (NSYN+ATR),BC
        LD BC,XATR
        LD (DESC+ATR),BC
        CALL GETATR
;COPY HDR BLKS TO XX,YY
        LD HL,(XATR)
        LD DE,XX
        LD BC,11
        LDIR
        LD HL,(YATR)
        LD DE,YY
        LD BC,11
        LDIR
        LD HL,(ZATR)
        LD DE,ZZ
        LDI
        LDI
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
STORZ:
;ALLOCATES NODE FOR BOOLEAN RESULT
        ;OF RELATIONAL FUNCTION.
;SEND TYP,SPC,IDX TO ALLOC
        LD A,BOOLN
        LD (TYP+ZZ),A
        LD BC,1
        LD (SPC+ZZ),BC
        LD DE,(ZATR)
        LD HL,ZZ
        LD BC,5
        LDIR
        CALL ALOSYN
        EX DE,HL
        LD BC,6
        LDIR
        LD IX,(ZZ+FST)
        LD A,(RESTYP)
        LD (IX),A
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
COMPAR:
;COMPARE X,Y
;RETURNS MINUS (SIGN FLAG SET) IF X<Y,
;ZERO (Z FLAG SET) IF X=Y, OR
;>0 (SIGN SET, Z NOT SET) IF X>Y
;FOR ATOMS: NIL<NUMBER<BOOLN<SYMBL<SEQNC
;ATOM<SEQUENCE
        CALL SAVREG
;IF TYP'S NOT EQUAL, WE ARE DONE
        LD A,(XX+TYP)
        LD BC,(YY+TYP)
        CP C
        JP NZ,CP.40
;BOTH OF SAME TYPE, COMPARE IN DETAIL
        LD IX,(XX+ADR)
        LD IY,(YY+ADR)
        AND OFOH
        CP NUMBR
        JP Z,CP.NUM
;
        CP SYMBL
        JP Z,CP.SYM
;
        CP SEQNC
        JP Z,CP.SEQ
;TYP NOT RECOGNIZABLE, RETURN ERROR MSG
        JP      CP.50
;
CP.NUM:
        CALL CMPNUM
        JP CP.40
;
CP.SYM:
        CALL CMPSYM
        JP CP.40
;
CP.SEQ:
        CALL CMPSEQ
;
CP.40:  CALL RSTREG
        RET
;
CP.50:  LD      DE,$CP.50
        CALL    PRLINE
        JP      CP.40
;
$CP.50: DB 'UNKNOWN TYPS: CANT COMPARE $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CMPNUM:
        CNT     EQU     NXT+2
;COMPARES 2 FIXED PT NUMBERS
        CALL SAVREG
;
; CHECK FOR NEGATIVE NUMBERS
;
        LD      A,(XX+TYP)      ; 2/15
        CP      NEGFXP          ; 2/15
        JP      NZ,CP.2         ; 2/15 BYPASS IF NEG
        LD      A,(IX+CNT)      ; 2/15 COMPARE WHL #'S
        CP      (IY+CNT)        ; 2/15
        JP      Z,CP.3          ; 2/15 LENGTHS EQUAL
        JP      M,CP.1          ; 2/15
        LD      A,0             ; 2/15
        CP      1               ; 2/15 SET FLAGS ACCORD.
        JP      CPN.10          ; 2/15 EXIT IF -Y<-X
CP.1:   LD      A,1             ; 2/15
        CP      0               ; 2/15 SET FLAGS ACCORD.
        JP      CPN.10          ; 2/15
```

```
;COMPARE # OF WHL DIGITS
CP.2:   LD A,(IX+CNT)
        CP (IY+CNT)
        JP NZ,CPN.10
;IF LENGTHS SAME, COMPARE EACH BYTE
;SAVE # OF DEC BYTES IN HL
CP.3:   LD L,(IX+CNT+1)
        LD H,(IY+CNT+1)
;MOVE IX,IY UP TO DATA
        LD C,(IX+CNT)
        LD DE,7
        ADD IX,DE
        ADD IY,DE
;LOOP TO COMPARE WHL BYTES
CPN.1:  DEC C
        JP M,CPN.2
        LD      A,(XX+TYP)        ; 2/15
        CP      NEGFXP            ; 2/15 CHECK FOR NEG #
        JP      NZ,CP.5           ; 2/15 BYBASS FOR POS #
        LD      A,(IX)            ; 2/15
        CP      (IY)              ; 2/15
        JP      Z,CP.6            ; 2/15 BYTES ARE =
        JP      M,CP.4            ; 2/15 X>Y
        LD      A,0               ; 2/15
        CP      1                 ; 2/15
        JP      CPN.10            ; 2/15
CP.4:   LD      A,1               ; 2/15
        CP      0                 ; 2/15
        JP      CPN.10            ; 2/15
CP.5:   LD A,(IX)
        CP (IY)
        JP NZ,CPN.10
CP.6:   INC IX
        INC IY
        JP CPN.1
CPN.2:
;WHL PARTS IDENTICAL, COMPARE DEC PARTS
        LD A,L
        CP H
        LD C,H
        JP P,CPN.3
        LD C,L
CPN.3:  DEC C
        JP M,CPN.4
        LD      A,(XX+TYP)                ; 2/15
        CP      NEGFXP                    ; 2/15
        JP      NZ,CP.8                   ; 2/15
        LD      A,(IX)                    ; 2/15
        CP      (IY)                      ; 2/15
        JP      Z,CP.9                    ; 2/15
        JP      M,CP.7                    ; 2/15
        LD      A,0                       ; 2/15
        CP      1                         ; 2/15
        JP      CPN.10                  ; 2/15
CP.7:   LD      A,1                     ; 2/15
        CP      0                       ; 2/15
        JP      CPN.10                  ; 2/15
CP.8:   LD A,(IX)
        CP (IY)
        JP NZ,CPN.10
CP.9:   INC IX
        INC IY
        JP CPN.3
CPN.4:  LD      A,(XX+TYP)              ; 2/15
        CP      NEGFXP                  ; 2/15
        JP      NZ,CP.11                ; 2/15
        LD      A,L                     ; 2/15
        CP      H                       ; 2/15
        JP      M,CP.10                 ; 2/15
        LD      A,0                     ; 2.15
        CP      1                       ; 2/15
        JP      CPN.10                  ; 2/15
CP.10:  LD      A,1                     ; 2/15
        CP      0                       ; 2/15
        JP      CPN.10                  ; 2/15
CP.11:  LD      A,L                     ; 2/15
        CP      H                       ; 2/15
CPN.10: CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CMPSYM:
;OBJECTS ARE SYMBOL'S
;COMPARE LENGTHS FIRST, THEN EACH ITEM
        CALL SAVREG
        LD HL,(XX+SPC)
        LD BC,(YY+SPC)
        AND 0
        SBC HL,BC
        JP NZ,CPS.10
;MOVE IX,IY UP TO DATA
        LD IX,(FST+XX)
        LD IY,(FST+YY)
CPS.1:  DEC C
        JP P,CPS.2
        DEC B
        JP M,CPS.3
CPS.2:
        LD A,(IX)
        CP (IY)
        JP NZ,CPS.10
        INC IX
        INC IY
        JP CPS.1
CPS.3:  CP A
CPS.10: CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
CMPSEQ:
;COMPARE EACH OBJECT IN 2 SEQNC'S
        CALL SAVREG
;FIRST COMPARE THEIR LENGTHS
        LD HL,(XX+SPC)
        LD BC,(YY+SPC)
        AND 0
        SBC HL,BC
        JP NZ,CPQ.10
;MUST COMPARE PAIRS OF OBJECTS
;SET UP LOOP TO STEP THROUGH SEQNC'S
        SRA B
        RR C
        LD IX,(XX+FST)
        LD IY,(YY+FST)
CPQ.1:  DEC C
        JP P,CPQ.2
        DEC B
        JP M,CPQ.3
CPQ.2:  PUSH BC
;FETCH 1 OBJ FROM X,Y EACH
        LD L,(IX)
        LD H,(IX+1)
        LD (IDX+HDR),HL
        CALL FETCH
        LD HL,IDX+HDR
        LD DE,XX
        LD BC,11
        LDIR
        LD L,(IY)
        LD H,(IY+1)
        LD (IDX+HDR),HL
        CALL FETCH
        LD HL,IDX+HDR
        LD DE,YY
        LD BC,11
        LDIR
;CALL COMPAR
        CALL COMPAR
        LD BC,2
        ADD IX,BC
        ADD IY,BC
        POP BC
        JP NZ,CPQ.10
        JP CPQ.1
CPQ.3:  CP A
CPQ.10:
        CALL RSTREG
        RET
;
CLUP1:  DW      0
TEMP:   DW      0
XATR:   DW      0
YATR:   DW      0
ZATR:   DW      0
XX:     DS      12
YY:     DS      12
ZZ:     DS      12
        END                     ; END OF RELN
```

```
;
; 5 AUG 83 - ORIGINAL
; 10 OCT 83 - REMOVED SYNTAX ERRORS
; 18 NOV 83 - MODIFIED COLECT
; 23 NOV 83 - ADDED BCCHECK
; 29 NOV 83 - MODIFIED GC
; 12 JAN 84 - REMOVED UNNECESSARY PRINT STATEMENTS
;
        TITLE STOR A/O 12 JAN 84
;
GLOBAL ALLOC,FETCH
GLOBAL GETNOD
GLOBAL SLIDE,COLECT
EXTERNAL PRLINE,SAVREG,RSTREG
EXTERNAL HDR,PTR,INHSTK,NODLST,NODES
                .XLIST
        MACLIB  EQATMO
          EQUATES
                .LIST
;
        FAIL    EQU         0
        SUCC    EQU         1
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
FETCH:
        CALL SAVREG
;LOOK UP ADDR OF NODE WITH INDEX (IDX+HDR)
        CALL GETADR
        JP Z,FET.8
;IX = ADDR OF NODE
        LD IX,(ADR+HDR)
;GET NODE TYPE
        LD A,(IX+TYP)
        LD (TYP+HDR),A
;COMPUTE LAST ADDR
        LD E,(IX+NXT)
        LD D,(IX+1+NXT)
        LD HL,(ADR+HDR)
        ADD HL,DE
;FOL = ADDR OF FOLLOWING NODE
        LD (FOL+HDR),HL
        DEC HL
        LD (LST+HDR),HL
;COMPUTE NODE'S DATA SPACE
        EX DE,HL
        LD DE,5
        AND 0
        SBC HL,DE
        LD (SPC+HDR),HL
;COMPUTE FIRST ADDR
        ADD IX,DE
        LD (FST+HDR),IX
        CALL RSTREG
        RET
FET.8:  LD HL,TYP+HDR
        LD (HL),0
        LD BC,8
        LD DE,TYP+HDR
        INC DE
        LDIR
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
ALLOC:  ;ALLOCATES STORAGE SPACE TO NODE(IDX)
        CALL SAVREG
  ;
ALC.1:
        LD HL,(SPC+HDR) ;TOTAL LENGTH
        LD BC,5         ;=SPC+5
        ADD HL,BC
        LD BC,(FREE+PTR);ADD LENGTH
        ADD HL,BC       ;TO FREE TO SEE
        AND 0           ;IF ENOUGH STORAGE
        LD BC,(LAST+PTR);SPACE LEFT
        SBC HL,BC       ;FREE+SPC-LAST>0?
        JP M,ALC.2      ;IF NOT, ENOUGH SPACE
;IF NOT ENOUGH SPACE, COLECT GARBAGE
        LD HL,IDX+HDR
        LD DE,SAVNEW
        LD BC,5
        LDIR
        CALL COLECT
        LD HL,SAVNEW
        LD DE,IDX+HDR
        LD BC,5
        LDIR
        LD A,(GCSUCC)   ;TEST GC FLAG
        CP SUCC         ;IF SUCC
        JP Z,ALC.1      ;TEST FREE SPACE AGAIN
        LD DE,$ALC.1
        CALL PRLINE
        CALL RSTREG
        RET
;
ALC.2:
;SET IX TO POINTER INTO NODE LIST
        CALL LOOKUP
        LD HL,(FREE+PTR);HL=1ST ADR OF NODE
        LD (IX),L       ;STORE FREE IN
        LD (IX+1),H     ;IN LODLST(NODE)
;HL=NODE ADDR
        LD (ADR+HDR),HL
;STORE NODE'S IDX AND TYP
```

```
        LD IX,(ADR+HDR)
        LD DE,(IDX+HDR)
        LD (IX+IDX),E
        LD (IX+1+IDX),D
        LD A,(TYP+HDR)
        LD (IX+TYP),A
;ADD 5 TO SPC TO GET NXT
        LD BC,5
        LD DE,(SPC+HDR)
        LD HL,0
        ADD HL,BC
        ADD HL,DE
        LD (IX+NXT),L
        LD (IX+1+NXT),H
;STORE FST,LST,FOL IN HDR BLCK
        ADD IX,BC
        LD (FST+HDR),IX
        ADD IX,DE
        LD (FOL+HDR),IX
        DEC IX
        LD (LST+HDR),IX
;UPDATE (FREE)
        LD BC,(FREE+PTR)
        ADD HL,BC
        LD (FREE+PTR),HL
        CALL RSTREG
        RET
;
$ALC.1: DB 'NOT ENOUGH FREE SPACE $'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GETADR:
;LOOKS UP ADDR OF NODE WITH INDEX IDX
;STORES NODE ADDR IN (ADR+HDR)
        CALL SAVREG
        CALL LOOKUP
        LD L,(IX)
        LD H,(IX+1)
        LD (ADR+HDR),HL
        OR 0
        LD BC,0
        SBC HL,BC
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;ROUTINES TO GET AND PUT NODE INDICES
;
GETNOD:
        CALL SAVREG
GET.0:
        LD IX,NODLST-4
        LD HL,0
        LD BC,NUMNOD
        LD DE,4
GET.1:
        DEC C
        JP NZ,GET.2
        DEC B
        JP M,GET.3
GET.2:
        ADD IX,DE
        INC HL
        LD A,(IX)
        CP NILIDX
        JP NZ,GET.1
        LD A,(IX+1)
        CP NILIDX
        JP NZ,GET.1
        LD (IX),TKNIDX
        LD (IX+1),TKNIDX
        LD (IDX+HDR),HL
        CALL RSTREG
        RET
GET.3:
        CALL COLECT
        LD A,(GCSUCC)
        CP SUCC
        JP Z,GET.0
        LD DE,$MSGT2
        CALL PRLINE
        CALL RSTREG
        RET
$MSGT2: DB ': NEED MORE NODES! '
        DB  ODH,OAH,'$'
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
LOOKUP:
;LOOKS UP NODE SLOT IN NODLST, IX PTS TO SLOT
        LD DE,(IDX+HDR)
        LD IX,NODLST-4
        ADD IX,DE
        ADD IX,DE
        ADD IX,DE
        ADD IX,DE
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
COLECT:
;ROUTINES TO MARK, RELEASE AND COLLECT
;NODES WHICH ARE NO LONGER NEEDED
```

```
;
        CALL SAVREG
;MARK ALL NODES BELOW TOP OF INHSTK AND THEIR CHILDREN
        CALL MRKSTK
;RELEASE ALL UNMARKED NODES IN NODLST
        CALL RELEAS
;LET GARBAGE COLLECTOR COMPACT
;RELEASED STORAGE SPACE
        CALL GC
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
MRKSTK:
;MARKS ALL NODES BELOW TOP, AND ALL CHILDREN
;FIRST DEMARK ALL
        LD BC,NUMNOD
        LD IX,NODLST
        LD DE,4
DEMK.1: DEC C
        JP NZ,DEMK.2
        DEC B
        JP M,DEMK.3
        JP Z,DEMK.3
DEMK.2:
        LD (IX+2),0
        ADD IX,DE
        JP DEMK.1
DEMK.3:
        LD A,1
        LD (MARK),A
;
;STORE CURRENT BAS AT TOP
        LD BC,(BAS+PTR)
        LD IX,(TOP+PTR)
        LD (IX),C
        LD (IX+1),B
        LD (HIBAS),IX
;STEP DOWN THRU FRAMES UNTIL BTM OF INHSTK
MKST.1: CALL SHFBAS
;NOW (LOBAS)=BAS OF CURRENT FRAME
;AND (NUMAT)=# OF ATTR. IN THE FRAME
        LD IX,(LOBAS)
        INC IX
        INC IX
;IX PTS TO FIRST DATA BYTE
        LD BC,(NUMAT)
        CALL BCNOTZ
        JP Z,MKST.2
;MARK BC NODES IN CURRENT FRAME
        CALL MARKER
MKST.2:
        LD HL,(LOBAS)
        LD (HIBAS),HL
;COMPARE IT TO INHSTK BTM
        LD DE,INHSTK
        AND 0
        SBC HL,DE
        RET M
        JP NZ,MKST.1
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
MARKER:
;MARKS BC NODES, WITH FIRST INDEX
;POINTED TO BY IX
MK.10:  CALL    BCCHECK                 ; 11/23
        JP      Z,MK.20                 ; 11/23
        DEC     BC                      ; 11/23
MK.11:  LD E,(IX)
        LD D,(IX+1)
        LD (IDX+HDR),DE
        DEC DE
        LD IY,NODLST
        ADD IY,DE
        ADD IY,DE
        ADD IY,DE
        ADD IY,DE
        LD A,(MARK)
        LD (IY+2),A
        LD A,(IY)
        CP TKNIDX
        JP NZ,MK.2
        LD A,(IY+1)
        CP TKNIDX
        JP Z,MK.14
MK.2:   CALL FETCH
        LD A,(TYP+HDR)
        CP SEQNC
        JP Z,MK.12
        CP STREM
        JP NZ,MK.14
;SEQ OR STR FOUND: MUST CALL MARKER RECURSIVELY
MK.12:  CALL SAVREG
        LD IX,(FST+HDR)
        LD BC,(SPC+HDR)
        CALL BCNOTZ
        JP Z,MK.13
        SRL B
        RR C
        CALL MARKER
MK.13:
        CALL RSTREG
MK.14:  INC IX
```

```
        INC IX
        JP MK.10
;ALL DONE
MK.20:
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SHFBAS:
;ROUTINE TO MAKE (LOBAS) PT TO NEXT
;BAS BELOW (HIBAS), (NUMAT)=# OF INDICES BETWEEN
        CALL SAVREG
        LD HL,(HIBAS)
        LD E,(HL)
        INC HL
        LD D,(HL)
        LD (LOBAS),DE
        DEC HL
        DEC HL
        DEC HL
        AND 0
        SBC HL,DE
        SRL H
        RR L
        LD (NUMAT),HL
        CALL RSTREG
        RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
RELEAS:
;RELEASES ANY UNMARKED NODES IN NODLST,
        CALL SAVREG
        LD BC,NUMNOD
        LD IX,NODLST
        LD A,(MARK)
        LD D,A
RLS.11: CALL    BCCHECK         ; 11/23 - ALL NODES PROCES
        JP      Z,RLS.13        ; 11/23 - HERE IF YES
        DEC     BC              ; 11/23
RLS.10:
        LD A,(IX+2)
        CP D
        JP Z,RLS.12
        LD A,NILIDX
        CP (IX)
        JP NZ,RLS.15
        CP (IX+1)
        JP Z,RLS.12
RLS.15:
        LD L,(IX)
        LD H,(IX+1)
        LD (HL),NILIDX
```

```
         INC HL
         LD (HL),NILIDX
         LD (IX),NILIDX
         LD (IX+1),NILIDX
RLS.12:  INC IX
         INC IX
         INC IX
         INC IX
         JP RLS.11
RLS.13:
         CALL RSTREG
         RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
SLIDE:
;ROUTINE TO SLIDE TOP-MOST ATTR FRAME
;DOWN OVER FRAME JUST BELOW IT
;RESETS BAS AND TOP
         CALL SAVREG
;SET IX TO TOP AND STORE BAS THERE
         LD IX,(TOP+PTR)
         LD BC,(BAS+PTR)
         LD (IX),C
         LD (IX+1),B
         LD (HIBAS),IX
;CALL SHFBAS TO GET # OF BYTES IN TOP FRAME
         CALL SHFBAS
         LD BC,(NUMAT)
         SLA C
         RL B
;SHIFT DOWN ONE MORE FRAME
         LD HL,(LOBAS)
         LD (HIBAS),HL
         CALL SHFBAS
;SAVE (LOBAS) AS NEW BAS
         LD DE,(LOBAS)
         LD (BAS+PTR),DE
;POINT HL,DE TO 1ST UPPER,LOWER DATA BYTES
         INC HL
         INC HL
         INC DE
         INC DE
         CALL BCNOTZ
         JP Z,SL.2
;MOVE FRAME DOWN
         LDIR
SL.2:
;DE IS NOW NEW TOP
         LD (TOP+PTR),DE
         CALL RSTREG
         RET
;
```

```
LOBAS:  DW      0
HIBAS:  DW      0
NUMAT:  DW      0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
GC:
;GARBAGE COLLECTOR
        CALL SAVREG
        LD IX,(BASE+PTR)
;THIS LOOP LOOKS FOR FIRST NIL NODE
;IF FREE REACHED, GC FAILS
GC.1:   CALL TESTFRE
        JP P,GC.9
;IF NIL NODE FOUND, GC SUCCESSFUL
        CALL TESTNIL
        JP Z,GC.2
        CALL NEXTIX
        JP GC.1
;SAVE COLLECTION PTR
GC.2:   LD (COLPTR),IX
;SET SUCCESS FLAG
        LD A,SUCC
        LD (GCSUCC),A
;THIS LOOP LOOKS FOR FIRST NON-NIL NODE
GC.21:  CALL NEXTIX
;IF FREE FOUND, QUIT
        CALL TESTFRE
        JP P,GC.8
        CALL TESTNIL
        JP NZ,GC.3
        JP GC.21
;NON-NIL FOUND, MOVE IT
GC.3:
;SET MOVE PTR
        LD (MOVPTR),IX
;RESET THE NODE'S ADDR
        LD DE,(COLPTR)
        LD C,(IX+IDX)
        LD B,(IX+IDX+1)
        LD IY,NODLST-4
        ADD IY,BC
        ADD IY,BC
        ADD IY,BC
        ADD IY,BC
        LD (IY),E
        LD (IY+1),D
        LD C,(IX+NXT)
        LD B,(IX+NXT+1)
;TEST THAT BC>0
        LD A,0
        CP C
        JP M,GC.31
        CP B
        JP P,GC.91
GC.31:
;MOVE BC BYTES FROM 'MOV' TO 'COL'
        LD HL,(MOVPTR)
        LDIR
;MOVE COL FORWARD
        LD (COLPTR),DE
;GO BACK TO LOOK FOR NEXT NON-NIL
        JP GC.21
;SET FREE TO LAST (COLPTR)
GC.8:   LD IX,(COLPTR)
        LD (FREE+PTR),IX
        JP GC.10
;GC FAILED, SEND A MSG
GC.9:   LD DE,$GC.2
        CALL    PRLINE          ; 1/12
        LD A,FAIL
        LD (GCSUCC),A
GC.10:  CALL    RSTREG          ; 1/12
        RET
GC.91:  LD DE,$GC.91
        CALL    PRLINE          ; 1/12
        JP      GC.10
;
$GC.2:  DB 'NO GARBAGE FOUND '
        DB ODH,OAH,'$'
$GC.91: DB 'NXT < = 0: ERROR! '
        DB ODH,OAH,'$'
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NEXTIX:         ;BC<NEXT(IX)
        LD C,(IX+NXT)
        LD B,(IX+NXT+1)
        ADD IX,BC
        RET
;
TESTFRE:        ;TESTS IF IX = FREE
        PUSH IX
        POP  HL         ;HL=IX
        LD DE,(FREE+PTR)
        AND 0           ;CLEAR CARRY
        SBC HL,DE       ;COMPARE BY SUBTRACTION
        RET             ;TEST FOF NZ UPON RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
TESTNIL:        ;TEST IF IX IS NIL
        LD A,(IX+IDX)   ;TEST BOTH BYTES
        CP NILIDX               ;OF INDEX(IX)
        RET NZ
        LD A,(IX+IDX+1) ;AGAINST NIL
        CP NILIDX                   ;TEST FOR NZ
        RET                 ;UPON RETURN
;
BCNOTZ:
        CALL SAVREG
        LD HL,0
        OR 0
        SBC HL,BC
        CALL RSTREG
        RET
BCCHECK: XOR    A                   ; CLEAR ACCUMULATOR
        ADD     A,B
        JP      Z,BC1               ; CHECK C REGISTER
        RET                         ; ELSE RETURN
BC1:    XOR     A                   ; CLEAR ACCUMULATOR
        ADD     A,C
        RET
;
;
MARK:   DB      1
SAVNEW: DS      20H
COLPTR: DW      0
MOVPTR: DW      0
GCSUCC: DW      0
        END
;END OF STOR....................................
```

```
;
; 7 OCT 83
;
; ALL OF THE MACROS FOR ZBADJR ARE IN THIS FILE
;
;THE FRAME HANDLING FUNCTIONS...
;SET A NEW BAS, OBAS
SETINH  MACRO
        CALL SETINH
   ENDM
;
;STACK LIST OF ATTRS. ONTO CURRENT FRAME
STKINH  MACRO   VAR
        IRP     P,<VAR>
                LD BC,P
                CALL STKINH
        ENDM
   ENDM
;
;SHORTHAND FOR SETINH,STKINH
INHER   MACRO   VAR
        CALL SETINH
        STKINH <VAR>
  ENDM
;
;SHORTHAND FOR CALL RSTINH
DISINH  MACRO
        CALL RSTINH
  ENDM
;
;RESET BAS,OBAS TO PREVIOUS VALUES
RSTINH  MACRO
        CALL RSTINH
   ENDM
;
SETBAS  MACRO
        CALL SETBAS
    ENDM
;
RSTBAS  MACRO
        CALL RSTBAS
    ENDM
;
;DEFINE A LIST OF LOCAL ATTRIBUTES
DEFLOC  MACRO   VAR
        IRP P,<VAR>
                CALL DEFLOC
        ENDM
   ENDM
;
QUES    MACRO   B,NXTALT
        INHER <B>
        LD HL,NXTALT
```

```
            CALL QUES
      ENDM
;
ENDALT      MACRO   ENDLINE
            JP ENDLINE
      ENDM
;
SLIDE       MACRO
            CALL SLIDE
      ENDM
;
BAND        MACRO   VAR
            INHER <VAR>
              CALL BAND
            DISINH
      ENDM
;
BOR         MACRO   VAR
            INHER <VAR>
              CALL BOR
            DISINH
      ENDM
;
BXOR        MACRO   VAR
            INHER <VAR>
              CALL BXOR
            DISINH
      ENDM
;
BNOT        MACRO   VAR
            INHER <VAR>
              CALL BNOT
            DISINH
      ENDM
;
ATOM?       MACRO   VAR
            INHER <VAR>
              CALL ATOM?
            DISINH
      ENDM
;
NIL?        MACRO   VAR
            INHER <VAR>
              CALL NIL?
            DISINH
      ENDM
;
SYMBOL?     MACRO   VAR
            INHER <VAR>
              CALL SYMBOL?
            DISINH
      ENDM
;
```

```
NUMBER? MACRO   VAR
        INHER <VAR>
          CALL NUMBER?
        DISINH
  ENDM
;
BOOLEAN?        MACRO   VAR
        INHER <VAR>
          CALL BOOLEAN?
        DISINH
  ENDM
;
EMPTY?  MACRO   VAR
        INHER <VAR>
          CALL EMPTY?
        DISINH
  ENDM
;
SEQUENCE?       MACRO   VAR
        INHER <VAR>
          CALL SEQUENCE?
        DISINH
  ENDM
;
FINITE? MACRO   VAR
        INHER <VAR>
          CALL FINITE?
        DISINH
  ENDM
;
STREAM? MACRO   VAR
        INHER <VAR>
          CALL STREAM?
        DISINH
  ENDM
;
DRY?    MACRO   VAR
        INHER <VAR>
          CALL DRY?
        DISINH
  ENDM
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
NUM     MACRO   X
;MACRO FOR IMMEDIATE NUMERIC CONSTANT FUNCTION
;FOR EXAMPLE: "NUM <' +12.34'>" MAKES A
;NUMBER NODE AND PUSHES ITS INDEX ONTO THE
;INHERITED ATTRIBUTE STACK
        LOCAL EN
        JP EN
        DB '#'
        DB X
```

```
EN:
        CALL NUMIMM
ENDM
;
SYM     MACRO   X
;MACRO FOR IMMEDIATE STRING FUNCTION
;FOR EXAMPLE: "SYM <'ABCD'>" CREATES A SYMBL
;NODE AND PUSHES ITS INDEX ONTO THE
;INHERITED ATTRIBUTE STACK
        LOCAL ES
        JP ES
        DB '#'
        DB X
ES:
        CALL SYMIMM
ENDM
;
SL      MACRO   X,I
;DOES SELECT OF I'TH ELEMENT OF SEQNC X
        STKINH X
        LD BC,I
        CALL SELIMM
  ENDM
;
SR      MACRO   X,I
;DOES SER OF I'TH ELEMENT FROM END OF X
        STKINH X
        LD BC,I
        CALL SERIMM
  ENDM
;
CONS    MACRO
        CALL SETINH
  ENDM
;
CLSCON  MACRO
        CALL CONIMM
  ENDM
;
CONCAT  MACRO
        CALL SETINH
  ENDM
;
CLSCAT  MACRO
        CALL CATIMM
  ENDM
;
HEAD    MACRO   X
        STKINH <X>
        CALL HEAD
  ENDM
;
TAIL    MACRO   X
```

```
          STKINH <X>
          CALL TAIL
   ENDM
;
MERGE     MACRO
;MERGES   SEQNC'S X,,,Y INTO SEQNC Z
          SETINH
   ENDM
;
CLSMER    MACRO
          CALL MERIMM
   ENDM
;
ID        MACRO   VAR
          INHER <VAR>
            CALL ID
          DISINH
   ENDM
;
SEQSTR    MACRO   VAR
          INHER <VAR>
            CALL SEQSTR
          DISINH
   ENDM
;
STRSEQ    MACRO   VAR
          INHER <VAR>
            CALL STRSEQ
          DISINH
   ENDM
;
SYMSEQ    MACRO   VAR
          INHER <VAR>
            CALL SYMSEQ
          DISINH
   ENDM
;
SEQSYM    MACRO   VAR
          INHER <VAR>
            CALL SEQSYM
          DISINH
   ENDM
;
SEQNUM    MACRO   VAR
          INHER <VAR>
            CALL SEQNUM
          DISINH
   ENDM
;
NUMSYM    MACRO   VAR
          INHER <VAR>
            CALL NUMSYM
          DISINH
   ENDM
;
RV        MACRO   VAR
          INHER <VAR>
            CALL RV
          DISINH
   ENDM
;
DL        MACRO   VAR
          INHER <VAR>
            CALL DL
          DISINH
   ENDM
;
DR        MACRO   VAR
          INHER <VAR>
            CALL DR
          DISINH
   ENDM
;
SEL       MACRO   VAR
          INHER <VAR>
            CALL SEL
          DISINH
   ENDM
;
SER       MACRO   VAR
          INHER <VAR>
            CALL SER
          DISINH
   ENDM
;
TR        MACRO   VAR
          INHER <VAR>
            CALL TR
          DISINH
   ENDM
;
;MACROS FOR RELATIONAL FUNCTIONS
;
EQ?       MACRO   X
          INHER <X>
          CALL EQ
          DISINH
   ENDM
;
NE?       MACRO   X
          INHER <X>
          CALL NE
          DISINH
   ENDM
;
LT?       MACRO   X
```

```
                INHER <X>
                CALL LT
                DISINH
        ENDM
;
LE?             MACRO   X
                INHER <X>
                CALL LE
                DISINH
        ENDM
;
GT?             MACRO   X
                INHER <X>
                CALL GT
                DISINH
        ENDM
;
GE?             MACRO   X
                INHER <X>
                CALL GE
                DISINH
        ENDM
;
AD              MACRO   X
                INHER <X>
                CALL AD
                DISINH
        ENDM
;
SB              MACRO   X
                INHER <X>
                CALL SB
                DISINH
        ENDM
;
ML              MACRO   X
                INHER <X>
                CALL ML
                DISINH
        ENDM
;
DV              MACRO   X
                INHER <X>
                CALL DV
                DISINH
        ENDM
;
AB              MACRO   X
                INHER <X>
                CALL AB
                DISINH
        ENDM
;
```

```
NG       MACRO   X
         INHER <X>
         CALL NG
         DISINH
  ENDM
;
INT      MACRO   X
         INHER <X>
         CALL INT
         DISINH
  ENDM
;
MD       MACRO   X,M,Z
         INHER <X,M,Z>
           DEFLOC<4,5,6,7>
           INT <2,4>
           DV <1,4,5>
           INT <5,6>
           ML <4,6,7>
           SB <1,7,3>
         RSTINH
    ENDM
;
RDNUM    MACRO   X
         INHER <X>
         CALL RDNUM
         DISINH
  ENDM
;
PRNUM    MACRO   X
         INHER <X>
         CALL PRNUM
         DISINH
  ENDM
;
RDSYM    MACRO   X
         INHER <X>
         CALL RDSYM
         DISINH
  ENDM
;
PRSYM    MACRO   X
         INHER <X>
         CALL PRSYM
         DISINH
  ENDM
;
;
RDBUL    MACRO   X
         INHER <X>
         CALL RDBUL
         DISINH
  ENDM
```

```
;
PRBUL   MACRO   X
        INHER <X>
        CALL PRBUL
        DISINH
  ENDM
;MACROS FOR WHILE,APPLY-TO-ALL,INSERT
;
WHILE   MACRO   XZR,FN,ATR
        SETINH
          STKINH<XZR>
          CALL WHILE1
          STKWLD <ATR>
          LD HL,FN
          CALL WHILE2
        RSTINH
  ENDM
;
APPLYTOALL   MACRO      XZ,FN,ATR
        SETINH
         STKINH<XZ>
         CALL APPLY1
         STKWLD <ATR>
         LD HL,FN
         CALL APPLY2
        RSTINH
  ENDM
;
STKWLD  MACRO   VAR
        IRP P,<VAR>
        LD BC,P
        CALL STKWLD
        ENDM
  ENDM
;
INSERT  MACRO   ATR,FN
        INHER <ATR>
        LD HL,FN
        CALL INSERT
        DISINH
  ENDM
;DISK AND CONSOLE IO
RDCON   MACRO
        LD A,0
        CALL IOSEL
  ENDM
;
WRCON   MACRO
        LD A,1
        CALL IOSEL
  ENDM
;
RDOPEN  MACRO   FLNAME
```

```
          LOCAL CONT,NAME
          JP CONT
   NAME:  DB FLNAME
          DB      '.'
   CONT:
          LD HL,NAME
          LD A,2
          CALL IOSEL
   ENDM
;
WROPEN    MACRO   FLNAME
          LOCAL CONT,NAME
          JP CONT
   NAME:  DB FLNAME
          DB      '.'
   CONT:
          LD HL,NAME
          LD A,3
          CALL IOSEL
   ENDM
;
RDDSK     MACRO
          LD A,4
          CALL IOSEL
   ENDM
;
WRDSK     MACRO
          LD A,5
          CALL IOSEL
   ENDM
;
WRCLOS    MACRO
          LD A,6
          CALL IOSEL
   ENDM
;
;.............................................
```

```
;MACROS TO DEFINE OFFSETS
;
; 7 OCT 83
; 21 DEC 83 - MODIFIED BOOLEAN, TRUE, AND FALSE
;
EQUATES         MACRO
;TYPE CONSTANTS
        NIL     EQU     0A0H
        NUMBR   EQU     0C0H
        NEGFXP  EQU     0C1H
        POSFXP  EQU     0C2H
        FIXPT   EQU     0C3H
        NEGFLP  EQU     0C8H
        POSFLP  EQU     0C9H
        SYMBL   EQU     0D0H
        BOOLN   EQU     0D0H
        ATOM    EQU     0DFH
        SEQNC   EQU     0E0H
        NULL    EQU     0E1H
        STREM   EQU     0F0H
        DRY     EQU     0F1H
;
; PTR DISPLACEMENTS
        BASE    EQU        0
        FREE    EQU        2
        LAST    EQU        4
        FLGC    EQU        6
        COL     EQU        8
        MOV     EQU       10
        OBAS    EQU       12
        BAS     EQU       14
        TOP     EQU       16
        CONS    EQU       18
        LAS     EQU       20
;ATTRIBUTE PASSING PARAMETERS
        NINH    EQU 0   ;# OF INH ATTR
        NSYN    EQU 2   ;# OF SYN ATTR
        DESC    EQU 4   ;BLK FOR DESCRIPTORS
        BLKSIZ  EQU 16  ;SIZE OF DESC BLKS
;
; HDR DISPLACEMENTS
        IDX     EQU     0
        TYP     EQU     IDX+2
        NXT     EQU     TYP+1
        SPC     EQU     TYP+1
;OTHER NODE PARAMETERS
        ADR     EQU     SPC+2
        FST     EQU     ADR+2
        LST     EQU     FST+2
        FOL     EQU     LST+2
;
;GENERAL CONSTANTS
;
```

```
          TRUE     EQU      054H
          FALSE    EQU      046H
          NILIDX   EQU      0FFH
          TKNIDX   EQU      0
          WILD     EQU      0FFFFH
          BOOL?    EQU      0FEFEH
          NUMNOD   EQU      100H
          MAXSTOR  EQU      1000H
     ENDM
;
;
```

BIBLIOGRAPHY

Dixon, Robert D., "The BADJR Report." The FLITE Project. Wright State University. 1983

Franklin, Richard [illegible]., "ZBADJR: An Implementation of the BADJR Machine in Z80 Assembly Language." Wright State University. 1983.

Sloan, John L., "An Implementation of BADJR on the PDP-11." Wright State University. 1983.